SAND2017- 10322 O

FY17 ASC P&EM L2 Milestone 6016: Tri-Lab Co-Design Milestone: In-Depth Performance Portability Analysis of Improved Integrated Codes on Advanced Architecture

Executive Summary

Author: Robert Hoekstra, 09/22/2017

Introduction

This milestone is a tri-lab deliverable supporting ongoing Co-Design efforts impacting applications in the Integrated Codes (IC) program element Advanced Technology Development and Mitigation (ATDM) program element. In FY14, the trilabs looked at porting proxy application to technologies of interest for ATS procurements. In FY15, a milestone was completed evaluating proxy applications in multiple programming models and in FY16, a milestone was completed focusing on the migration of lessons learned back into production code development. This year, the co-design milestone focuses on extracting the knowledge gained and/or code revisions back into production applications.

Milestone Description

This milestone will exercise the IC or ATDM codes using new and emerging programming models, to demonstrate portability and analyze performance characteristics on the latest advanced architectures expected to become widely available in FY17. Performance analysis tools will be used to do an in-depth analysis of those same or similar production codes on available ASC AT platforms and test beds, represented by the significantly disparate architectures of ATS-1/Trinity and ATS-2/Sierra. This will advance ASC co-design efforts by impacting one or more of the following:

- Determining performance bottlenecks in new architectures
- Characterizing ASC applications (for example, memory footprints, network utilization, instruction mixes) so as to better inform proxy application development and hardware design space evaluations
- Evaluating the portability and performance of different data structures and data layouts for important physics motifs across multiple architectures
- Evaluating the effectiveness of hardware and/or software-based memory and data management strategies to deal with automating movement between levels of system memory and I/O hierarchies
- Evaluating the performance analysis tools in these environments to determine their strengths and weaknesses, which will be invaluable in feedback to the vendors

Completion Criteria: This milestone will be completed when:

1. Improvements to the mini-applications to more closely represent our workloads are demonstrated on test-bed platforms and/or production systems.
2. A report has been completed by the 3 labs detailing lessons learned both successes and failures in regard to improvements.
3. The milestone team has communicated appropriate findings to vendors associated with the platforms evaluated in the milestone.

Impact Statement

The work performed under this milestone will lead to a single, more powerful, voice from the labs to the vendors when demanding changes in the tools and the toolchain and will have a tangible impact on these relations moving forward. This is one of the first times that these tools have been applied to large applications and the results of these analyses will help improve both the applications themselves and the tools.

The work performed under this milestone should be a strong demonstration to vendors (and others) that while proxy-applications are a good starting point for studies, they are not enough and full applications need to be used as well since they oftentimes present new and tougher challenges to the tools.

One example of co-design with vendors that occurred during this milestone is the joint MPI work undertaken to alleviate increased resource pressure with larger number of ranks. The net result was a more usable MPI, which helps with all applications on the machine.
This work demonstrated that the problem size is crucial when analyzing applications and that bottlenecks sometimes shift with problem sizes. This is something that should be conveyed to the vendors.

Robert Hoekstra (01422)
Simon Hammond (01422)
Michael Lang (LANL)
Ben Bergen (LANL)
David Richards (LLNL)
Robert Neely (LLNL)

memorandum
Applied Computer Science Group

September 11, 2017

From: Sriram Swaminarayan, LANL (Contact)
Paul Henning, LANL
Tom Brunner, LLNL
Patrick Brantley, LLNL
Travis Fisher, SNL
Michael Heroux, SNL

To: Mike Lang, LANL
Rob Hoekstra, SNL
Rob Neely, LLNL

Subject: 2017 Tri-lab ASC ATDM/CSSE/IC Co-design Level 2 Milestone

Milestone Title: Tri-Lab Co-Design Milestone: In-Depth Performance Portability Analysis of Improved Integrated Codes on Advanced Architecture (#6016)

Review Committee Comments:

The review committee believes the tri-lab co-design team fully met the completion criteria for this milestone.

The work performed under this milestone will lead to a single, more powerful, voice from the labs to the vendors when demanding changes in the tools and the toolchain and will have a tangible impact on these relations moving forward.

The committee was impressed by the large body of work undertaken and the information collected and presented in the slides. The committee was also impressed by the broad collection of production codes that were used in this milestone – this is one of the first times that these tools have been applied to large applications and the results of these analyses will help improve both the applications themselves and the tools.

The work performed under this milestone should be a strong demonstration to vendors (and others) that while proxy-applications are a good starting point for studies, they are not enough and full applications need to be used as well since they oftentimes present new and tougher challenges to the tools.

One excellent example of co-design with vendors that occurred during this milestone is the joint MPI work undertaken to alleviate increased resource pressure with larger number of ranks. The net result was a more usable MPI, which helps with all applications on the machine.

This work demonstrated that the problem size is crucial when analyzing applications and that bottlenecks sometimes shift with problem sizes. This is something that should be conveyed to the vendors.

The committee found the discussion of worthiness of metrics useful and appropriate – if we cannot believe the output of the tools that we have, the hours (or days) spent collecting data are for naught. Some of the noteworthy examples presented included the interpretation of memory bandwidth reported in VTune and analysis of vector operations counts.

Finally, for the FY18 milestone, the committee would like to see more GPU results, stronger collaboration between the labs on metrics gathering, consistent metrics reporting across applications and labs, and inclusion of applications with memory accesses that have graph-like characteristics. We also recommend that the team share results with stakeholders, including experts that are knowledgeable in the relationship between the proxy application and the full application, early to ensure that proxy application results reflect expectations from the full application.

Sincerely,

Sriram Swaminarayan, on behalf of the committee.
sriram@lanl.gov
+1-505-667-8647

2017 Co-Design L2 Milestone Annotated Slides

Performance Tools and Ardra

David Richards
ASC Co-Design Project Deputy

August 30, 2017

LLNL-PRES-XXXXXX

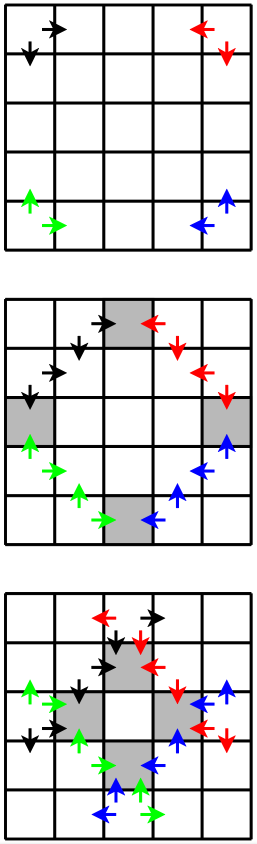
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

Ardra is LLNL’s next generation deterministic transport code

- Solves the neutral particle Boltzmann Transport Equation in 1D, 2D, and 3D
- RAJA refactoring is in progress
- Solution is neutron energy, direction of flight, and spatial distribution
  - We have MPI parallelism in each of these dimensions
- 4 major kernels
  - LTimes, LPlusTimes, Sweep, Fission

- Jez_scale_3d
  - Critical bare Pu sphere
  - Weak scale with mesh cells (increase resolution)
- Optimization Challenges
  - Vectorization
  - Difficult memory access patterns
  - Hoisting index calculation
  - Peeling indices while descending call stack complicates exposing parallelism
  - Wavefront (sweep) dataflow



Kripke, a proxy app for Ardra, was used to explore use of RAJA

- Led to new RAJA N-nested loop abstraction

RAJA style nested for-loop

```
RAJA::View vview(v_ptr,
                 make_perm_layout(ni,nj));
//...

RAJA::forallN< exec_policy, INDX, JNDX >(
               RangeSegment(1, ni),
               RangeSegment(0, nj),
               [=](INDX i, JNDX j) {

   vview(0, j) += vview(i, j);

});
```

- Lessons learned from Ardra refactoring are driving new improvements to RAJA

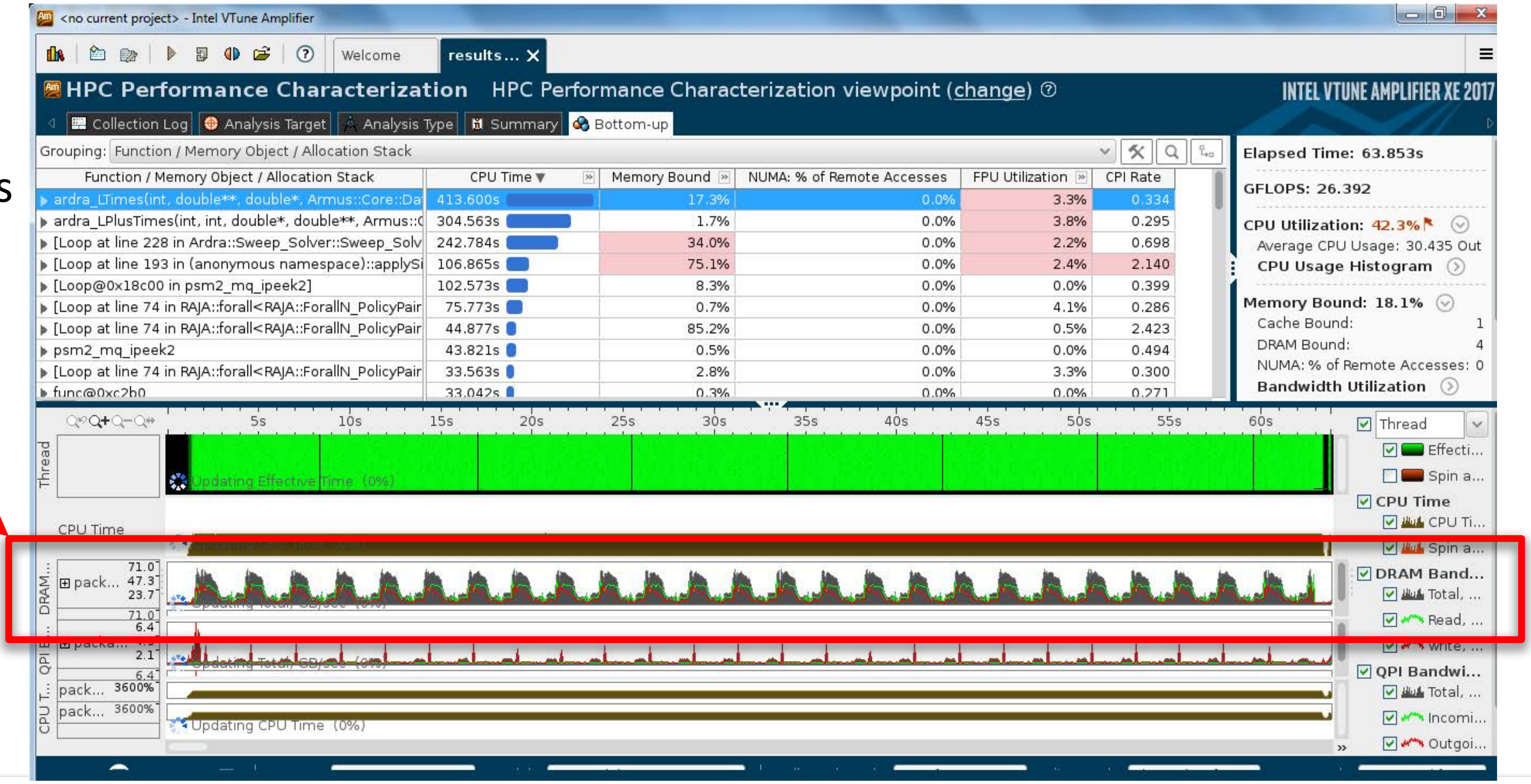
- LTimes and LPlusTimes kernels are streaming kernels of the form phi(nm,g,z) += ell(nm,d) * psi(d,g,z)
  - z is stride 1
  - ell(nm,d) is invariant in inner loop

- Previous Kripke studies found LTimes & LPlusTimes (and all other kernels) are memory bound on GPU, but not elsewhere (less than 25% of B/W).

- *Goal: Identify the bottleneck for these kernels on current platforms*

Intel VTune suggests Ardra is not B/W bound on CTS

DRAM B/W peaks at 50-60 GB/sec or ~50% of 130 GB/sec peak



<no current project> - Intel VTune Amplifier

HPC Performance Characterization — HPC Performance Characterization viewpoint (change)

Grouping: Function / Memory Object / Allocation Stack

Function / Memory Object / Allocation Stack	CPU Time ▼	Memory Bound	NUMA: % of Remote Accesses	FPU Utilization	CPI Rate
ardra_LTimes(int, double**, double*, Armus::Core::Da	413.600s	17.3%	0.0%	3.3%	0.334
ardra_LPlusTimes(int, int, double*, double**, Armus::(	304.563s	1.7%	0.0%	3.8%	0.295
[Loop at line 228 in Ardra::Sweep_Solver::Sweep_Solv	242.784s	34.0%	0.0%	2.2%	0.698
[Loop at line 193 in (anonymous namespace)::applySi	106.865s	75.1%	0.0%	2.4%	2.140
[Loop@0x18c00 in psm2_mq_ipeek2]	102.573s	8.3%	0.0%	0.0%	0.399
[Loop at line 74 in RAJA::forall<RAJA::ForallN_PolicyPair	75.773s	0.7%	0.0%	4.1%	0.286
[Loop at line 74 in RAJA::forall<RAJA::ForallN_PolicyPair	44.877s	85.2%	0.0%	0.5%	2.423
psm2_mq_ipeek2	43.821s	0.5%	0.0%	0.0%	0.494
[Loop at line 74 in RAJA::forall<RAJA::ForallN_PolicyPair	33.563s	2.8%	0.0%	3.3%	0.300
func@0xc2b0	33.042s	0.3%	0.0%	0.0%	0.271

Elapsed Time: 63.853s

GFLOPS: 26.392

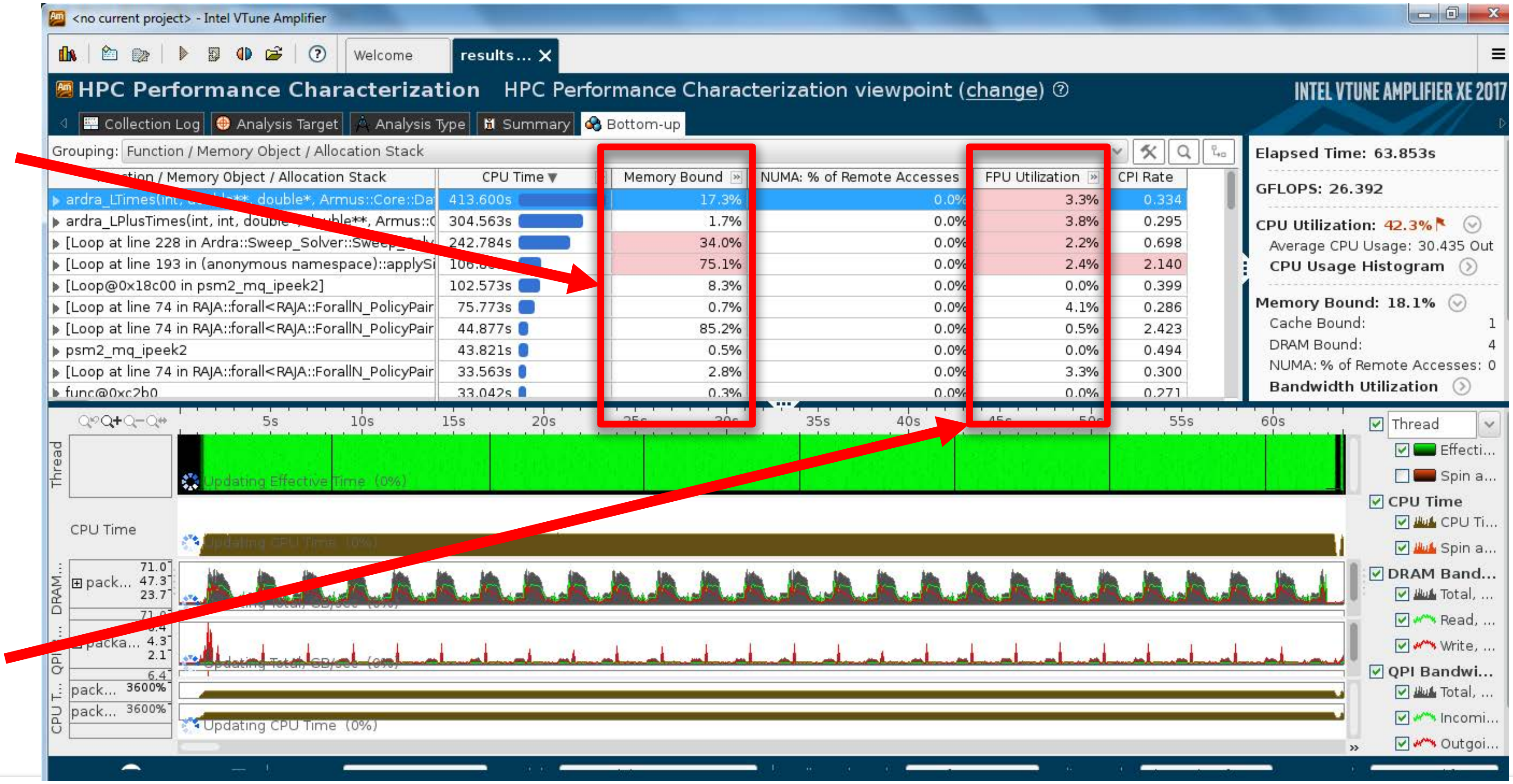
CPU Utilization: 42.3%
Average CPU Usage: 30.435 Out
CPU Usage Histogram

Memory Bound: 18.1%
Cache Bound: 1
DRAM Bound: 4
NUMA: % of Remote Accesses: 0
Bandwidth Utilization

Intel VTune suggests Ardra is not B/W bound on CTS

This diagnostic is confusing and not consistent with developer intuition

FPU usage is uniformly low



<no current project> - Intel VTune Amplifier

Welcome | results... ✕

HPC Performance Characterization   HPC Performance Characterization viewpoint (change) ⓘ   INTEL VTUNE AMPLIFIER XE 2017

Collection Log | Analysis Target | Analysis Type | Summary | Bottom-up

Grouping: Function / Memory Object / Allocation Stack

Function / Memory Object / Allocation Stack	CPU Time ▼	Memory Bound	NUMA: % of Remote Accesses	FPU Utilization	CPI Rate
▸ ardra_LTimes(int, double**, double*, Armus::Core::Da	413.600s	17.3%	0.0%	3.3%	0.334
▸ ardra_LPlusTimes(int, int, double*, double**, Armus::C	304.563s	1.7%	0.0%	3.8%	0.295
▸ [Loop at line 228 in Ardra::Sweep_Solver::Sweep_Solv	242.784s	34.0%	0.0%	2.2%	0.698
▸ [Loop at line 193 in (anonymous namespace)::applySi	106.8	75.1%	0.0%	2.4%	2.140
▸ [Loop@0x18c00 in psm2_mq_ipeek2]	102.573s	8.3%	0.0%	0.0%	0.399
▸ [Loop at line 74 in RAJA::forall<RAJA::ForallN_PolicyPair	75.773s	0.7%	0.0%	4.1%	0.286
▸ [Loop at line 74 in RAJA::forall<RAJA::ForallN_PolicyPair	44.877s	85.2%	0.0%	0.5%	2.423
▸ psm2_mq_ipeek2	43.821s	0.5%	0.0%	0.0%	0.494
▸ [Loop at line 74 in RAJA::forall<RAJA::ForallN_PolicyPair	33.563s	2.8%	0.0%	3.3%	0.300
▸ func@0xc2b0	33.042s	0.3%	0.0%	0.0%	0.271

Elapsed Time: 63.853s

GFLOPS: 26.392

CPU Utilization: 42.3%
Average CPU Usage: 30.435 Out
CPU Usage Histogram

Memory Bound: 18.1%
Cache Bound: 1
DRAM Bound: 4
NUMA: % of Remote Accesses: 0
Bandwidth Utilization

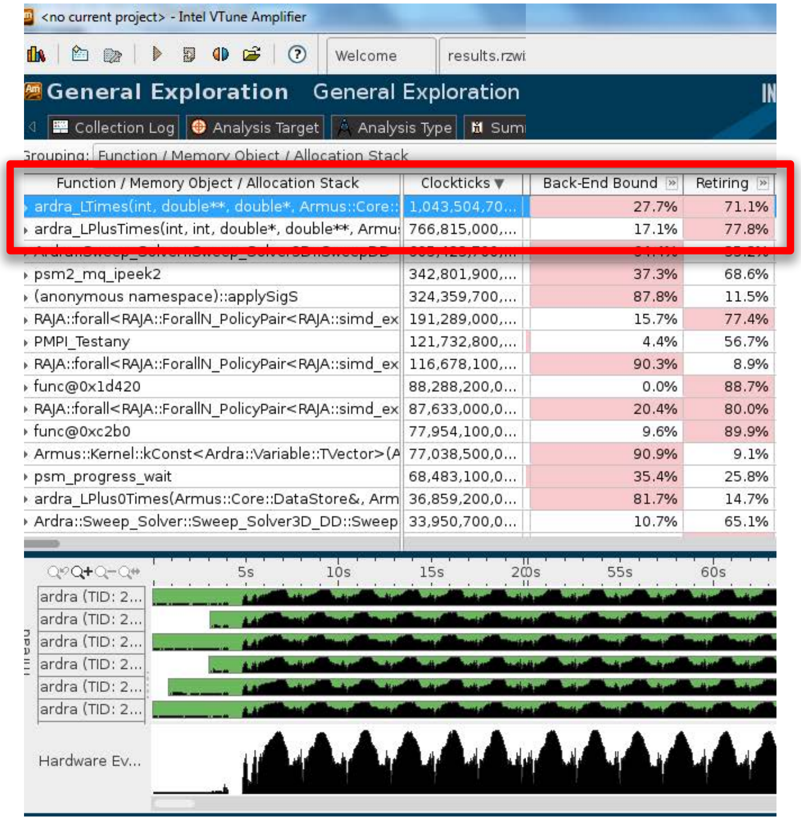
Root cause analysis requires deduction and guesswork

- Low flop rate throughout code (2.1% of peak)

- Nearly all flops are scalar instructions
  - VTune reports 12:1 ratio for scalar to vector flops
  - Allinea Map reports 6:1 ratio
  - VTune shows no vectorization in key kernels

- VTune shows high “Retiring” in LTimes and LPlusTimes

- Intel optimization manual says high retirement means you should vectorize
  - Allows more operations to complete per instruction



Function / Memory Object / Allocation Stack	Clockticks ▼	Back-End Bound	Retiring
ardra_LTimes(int, double**, double*, Armus::Core::	1,043,504,70...	27.7%	71.1%
ardra_LPlusTimes(int, int, double*, double**, Armu	766,815,000,...	17.1%	77.8%
Ardra::Sweep_Solver::Sweep_Solver3D::SweepDD	[illegible]	[illegible]	[illegible]
psm2_mq_ipeek2	342,801,900,...	37.3%	68.6%
(anonymous namespace)::applySigS	324,359,700,...	87.8%	11.5%
RAJA::forall<RAJA::ForallN_PolicyPair<RAJA::simd_ex	191,289,000,...	15.7%	77.4%
PMPI_Testany	121,732,800,...	4.4%	56.7%
RAJA::forall<RAJA::ForallN_PolicyPair<RAJA::simd_ex	116,678,100,...	90.3%	8.9%
func@0x1d420	88,288,200,0...	0.0%	88.7%
RAJA::forall<RAJA::ForallN_PolicyPair<RAJA::simd_ex	87,633,000,0...	20.4%	80.0%
func@0xc2b0	77,954,100,0...	9.6%	89.9%
Armus::Kernel::kConst<Ardra::Variable::TVector>(A	77,038,500,0...	90.9%	9.1%
psm_progress_wait	68,483,100,0...	35.4%	25.8%
ardra_LPlus0Times(Armus::Core::DataStore&, Arm	36,859,200,0...	81.7%	14.7%
Ardra::Sweep_Solver::Sweep_Solver3D_DD::Sweep	33,950,700,0...	10.7%	65.1%

After a lengthy saga, vectorization improved performance
(except when it didn't, and not the way we expected)

- RAJA abstractions are inhibiting vectorization
  - Refactoring and refinements are in progress
  - Manual unrolling and intrinsics were used to obtain vectorized code

- Performance improved:
  - Kernel times are significantly faster
  - But DRAM B/W changed very little

- Performance impact depends on problem size What's going on?
  - Small problem now operates at L2 B/W
  - Loads from L2 don't show up in DRAM B/W
  - Would blocking for L2 improve big problem performance?

Small Problem	scalar	vector
LTimes	15.53 sec	6.67 sec
LPlusTimes	22.43 sec	14.80 sec
DRAM B/W	60 GB/sec	68 GB/sec

Big Problem	scalar	vector
LTimes	84.75 sec	82.43 sec
LPlusTimes	101.38 sec	99.37 sec
DRAM B/W	117 GB/sec	119 GB/sec

CTS summary slide?

- Problem size matters
- B/W may not be what you think it is
- Bottleneck identification is hard

Ardra performance on KNL is ~1/2 of Broadwell (node to node)

- Limited tool set available. Mostly timing breakdowns by function
  - Data from internal timers
  - Similar data from CrayPAT, Map, PAPI
- KNL performance is consistent with expectations for codes not specifically tuned for KNL
- Performance difference not uniformly distributed across kernels

Time in Seconds	Broadwell	KNL	Ratio
Total	60.2	117.6	1.95
Solve	59.4	99.4	1.67
Sweep	12.3	31.8	2.58
LTimes	23.8	34.7	1.46
LPlusTimes	13.4	16.1	1.20
Fission	5.3	9.4	1.77

Ardra does not appear to be B/W bound on KNL

By changing NUMA modes we can get some hints that some kernels are B/W bound (at least they are sensitive), but we can’t get at the details without better tools

Time in Seconds	W/O HBM	With HBM
Total	133	117.6
Solve	117.4	99.4
Sweep	31.0	31.8
LTimes	48.6	34.7
LPlusTimes	16.1	16.1
Fission	10.1	9.4

LTimes is sensitive, but not even close to a 4x speedup

And LPlusTimes shows no change?!

We hoped vectorization would improve KNL performance. Preliminary trials show only a 10% gain in LTimes.

Performance Co-Pilot helps provide more accurate memory bandwidth attribution

- Measuring memory B/W is challenging for multiple reasons
  - Counters are on memory controller (“uncore”)
  - Not associated with any processor, thread, or execution context
  - Must use some model to attribute memory use to code

- Tools benefit from information from the application domain
  - PCP allows users to read memory controller counters and build their own models

- Data shown here assumes SPMD execution across all cores.

	VTune Memory Bound	PCP DRAM R/W bandwidth
Sweep	34%	20.0/12.5 GB/s
LTimes	17.3%	45.4/24.5 GB/s
LPlusTimes	1.75%	12.9/4.5 GB/s
Fission		47.1/45.9 GB/s

Ardra on GPU is relatively immature, but improving rapidly

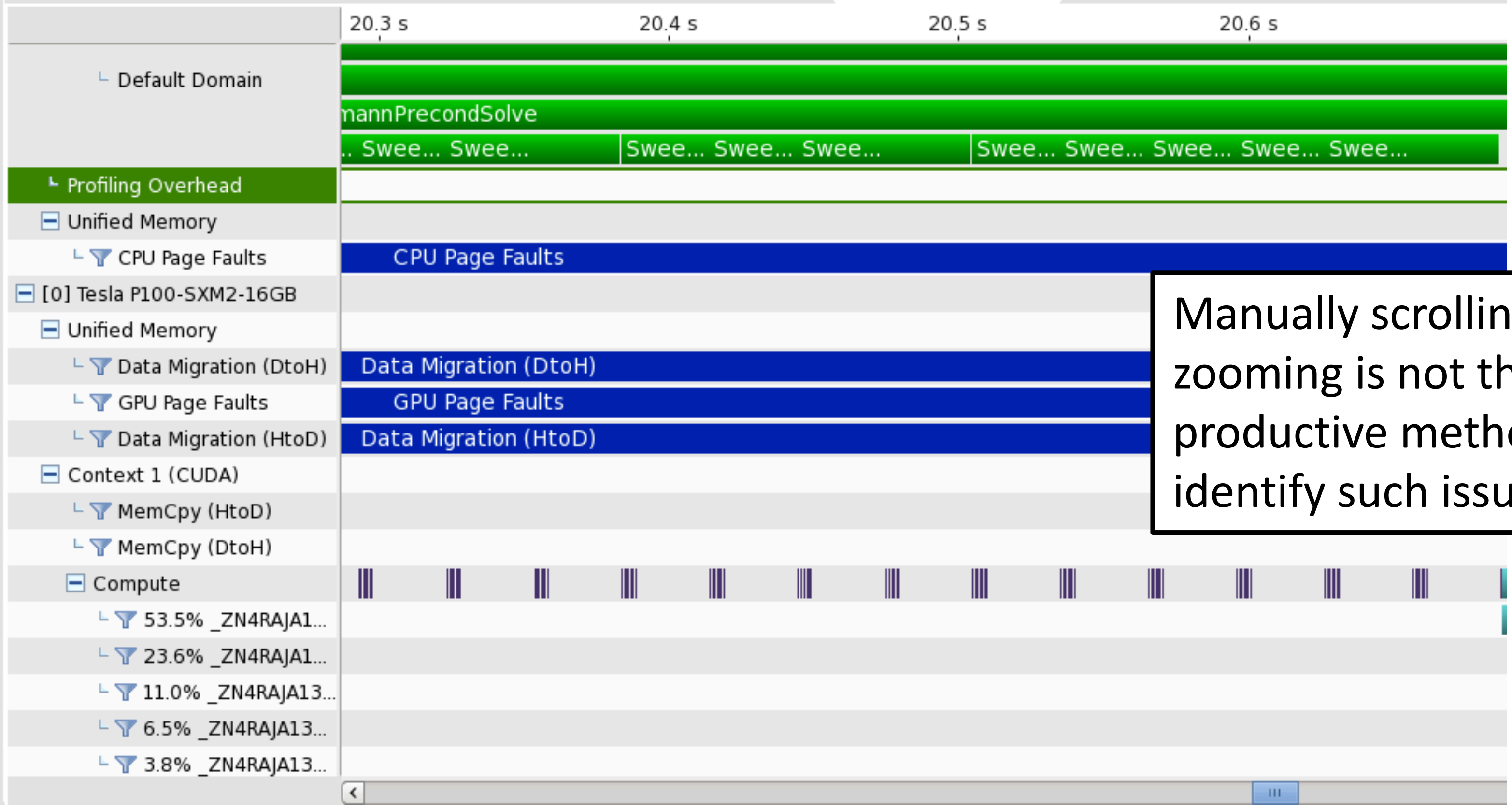
- Kripke helped show RAJA would work in Ardra. Now Ardra is driving improvements in RAJA design and implementation

- RAJA N-nested loops re-designed/re-implemented to fix disastrous performance
  - CHAI copy constructors were being called for each iteration of the inner loop

- Trials to date obtain only ~5 GB/sec out of 732 GB/sec peak B/W
  - Clearly not memory bound. Larger problems are needed
  - Problem size is limited by RAJA reduction implementation
  - Fixes are in progress

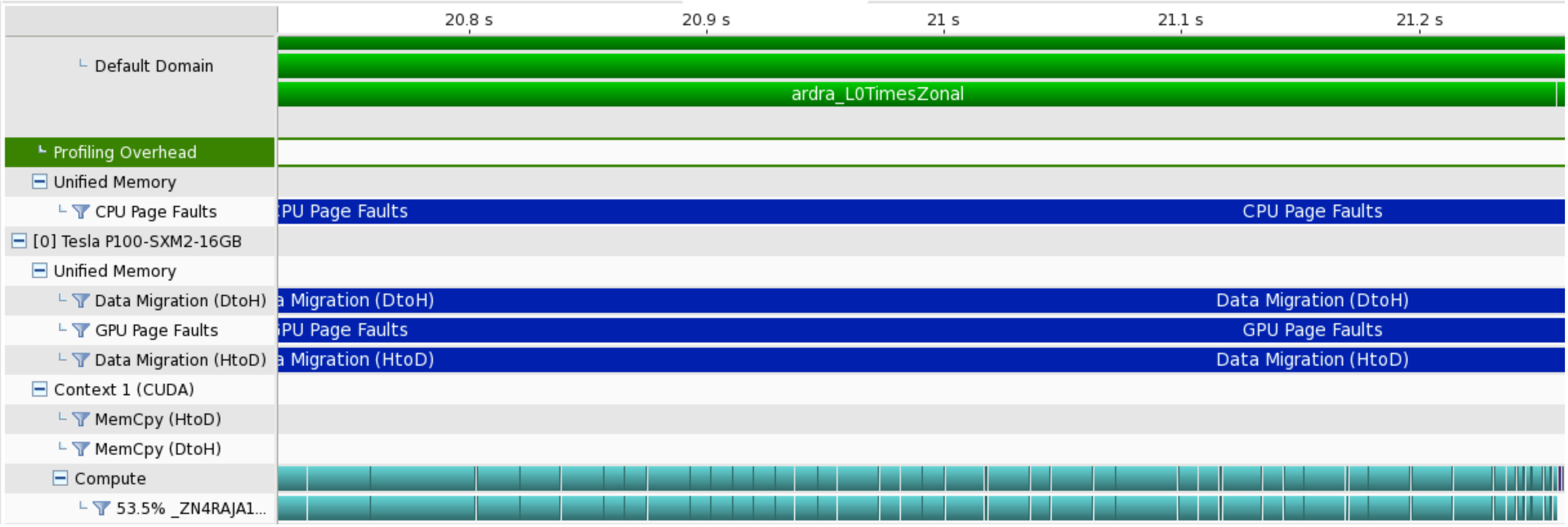
- MPI communication in sweep kernel is a major bottleneck
  - New algorithm has been demonstrated
  - Not yet implemented in Ardra

Nvprof shows sweep is dominated by non-compute time



Manually scrolling and zooming is not the most productive method to identify such issues

Compare sweep to L0Times that shows dense compute



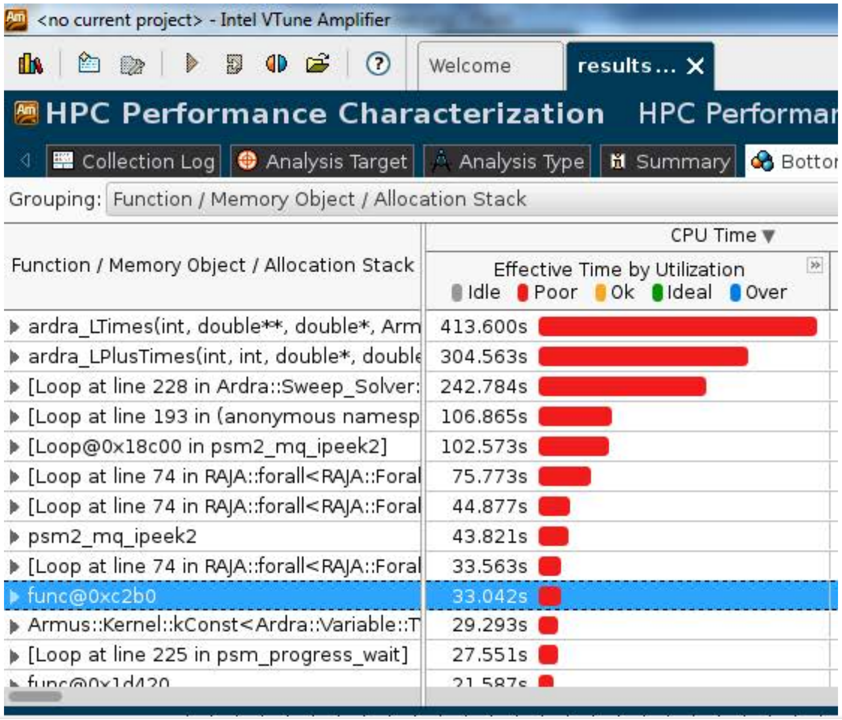
Other curiosities/frustrations: Tools don't agree on timings

Region	Region	Function
critSolver 97.5% 98.4% N/A 98% N/A		SigmaFTimes 10.7% 8.1% N/A N/A 1.0%
	Scattering 43.3% 55.6% 54.7% 64% 59.7%	LTimes 19.0% 23.9% 25.5% 40% 32.3%
		LPlusTimes 15.4% 20.0% 18.1% 23% 19.1%
		applySigS 7.5% 9.8% 11.1% N/A 8.3%
	Precond Solve 40.2% 30.7% N/A 35% N/A	MPI_Allreduce 20.9% 8.1% N/A N/A N/A
		Sweep_Solver3D 18.7% 21.9% 15.4% 20% 12.5%
Remainder 2.5% 1.6% N/A 2% N/A	Remainder 5.8% 5.6% N/A 1% N/A	Remainder 7.8% 9.2% N/A N/A N/A

Legend: HPCToolkit, MAP, GPerfTools, Built-In Timers, Open|SpeedShop

Other curiosities/frustrations: VTune recommendations can be less than helpful

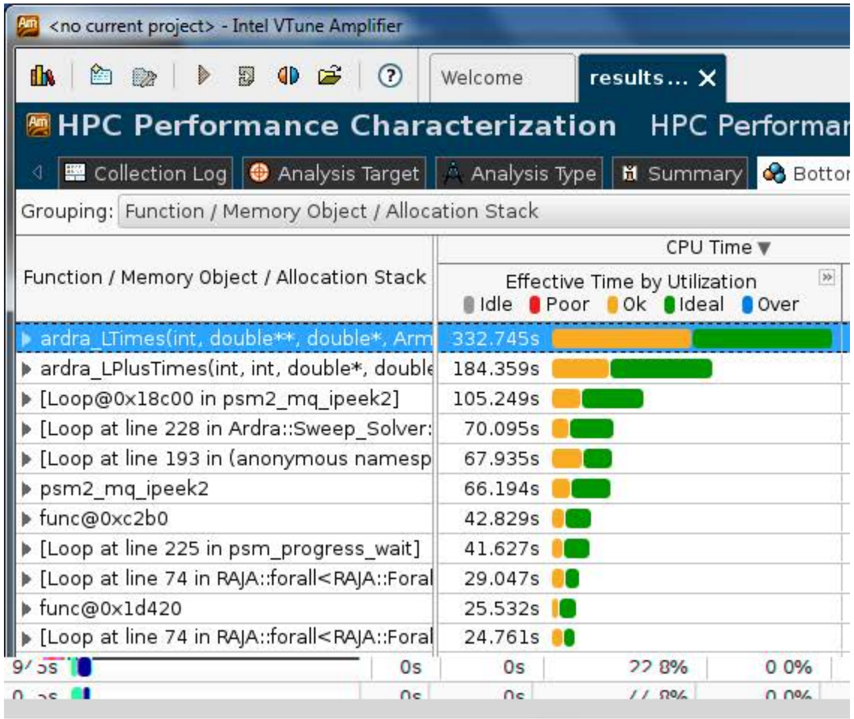
32 threads: 55.75 sec



Grouping: Function / Memory Object / Allocation Stack

Function / Memory Object / Allocation Stack	CPU Time ▼ Effective Time by Utilization ▮ Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over
ardra_LTimes(int, double**, double*, Arm	413.600s 🟥
ardra_LPlusTimes(int, int, double*, double	304.563s 🟥
[Loop at line 228 in Ardra::Sweep_Solver:	242.784s 🟥
[Loop at line 193 in (anonymous namesp	106.865s 🟥
[Loop@0x18c00 in psm2_mq_ipeek2]	102.573s 🟥
[Loop at line 74 in RAJA::forall<RAJA::Foral	75.773s 🟥
[Loop at line 74 in RAJA::forall<RAJA::Foral	44.877s 🟥
psm2_mq_ipeek2	43.821s 🟥
[Loop at line 74 in RAJA::forall<RAJA::Foral	33.563s 🟥
func@0xc2b0	33.042s 🟥
Armus::Kernel::kConst<Ardra::Variable::T	29.293s 🟥
[Loop at line 225 in psm_progress_wait]	27.551s 🟥
func@0x1d420	21.587s 🟥

64 threads: 55.67 sec



Grouping: Function / Memory Object / Allocation Stack

Function / Memory Object / Allocation Stack	CPU Time ▼ Effective Time by Utilization ▮ Idle 🟥 Poor 🟧 Ok 🟩 Ideal 🟦 Over
ardra_LTimes(int, double**, double*, Arm	332.745s 🟧🟩
ardra_LPlusTimes(int, int, double*, double	184.359s 🟧🟩
[Loop@0x18c00 in psm2_mq_ipeek2]	105.249s 🟧🟩
[Loop at line 228 in Ardra::Sweep_Solver:	70.095s 🟧🟩
[Loop at line 193 in (anonymous namesp	67.935s 🟧🟩
psm2_mq_ipeek2	66.194s 🟧🟩
func@0xc2b0	42.829s 🟧🟩
[Loop at line 225 in psm_progress_wait]	41.627s 🟧🟩
[Loop at line 74 in RAJA::forall<RAJA::Foral	29.047s 🟧🟩
func@0x1d420	25.532s 🟧🟩
[Loop at line 74 in RAJA::forall<RAJA::Foral	24.761s 🟧🟩

9/ 5s 0s 0s 22 8% 0 0%

Conclusion (needs to be refreshed/updated to latest results)

- CTS tools good enough for hypothesis generation
  - However, all guesses so far are wrong
  - Frequently end up looking at disassembled code

- Issues with misleading and confusing information
  - Incorrect or misleading hints are a huge waste of time

- Need to know algorithmic characteristics to fully understand performance
  - Especially memory reads/writes
  - Performance model and map to hardware are critical to understand if you're done optimizing

- Ardra is not B/W bound on any studied platform
  - GPU needs bigger problems
  - Bottleneck unclear on CTS and KNL

Lawrence Livermore National Laboratory

Backup slides showing that we have data for all of the metrics

Wall-Time Differences between Tools

Region	Region	Function
critSolver 97.5% 98.4% N/A 98% N/A		SigmaFTimes 10.7% 8.1% N/A N/A 1.0%
	Scattering 43.3% 55.6% 54.7% 64% 59.7%	LTimes 19.0% 23.9% 25.5% 40% 32.3%
		LPlusTimes 15.4% 20.0% 18.1% 23% 19.1%
		applySigS 7.5% 9.8% 11.1% N/A 8.3%
	Precond Solve 40.2% 30.7% N/A 35% N/A	MPI_Allreduce 20.9% 8.1% N/A N/A N/A
		Sweep_Solver3D 18.7% 21.9% 15.4% 20% 12.5%
Remainder 2.5% 1.6% N/A 2% N/A	Remainder 5.8% 5.6% N/A 1% N/A	Remainder 7.8% 9.2% N/A N/A N/A

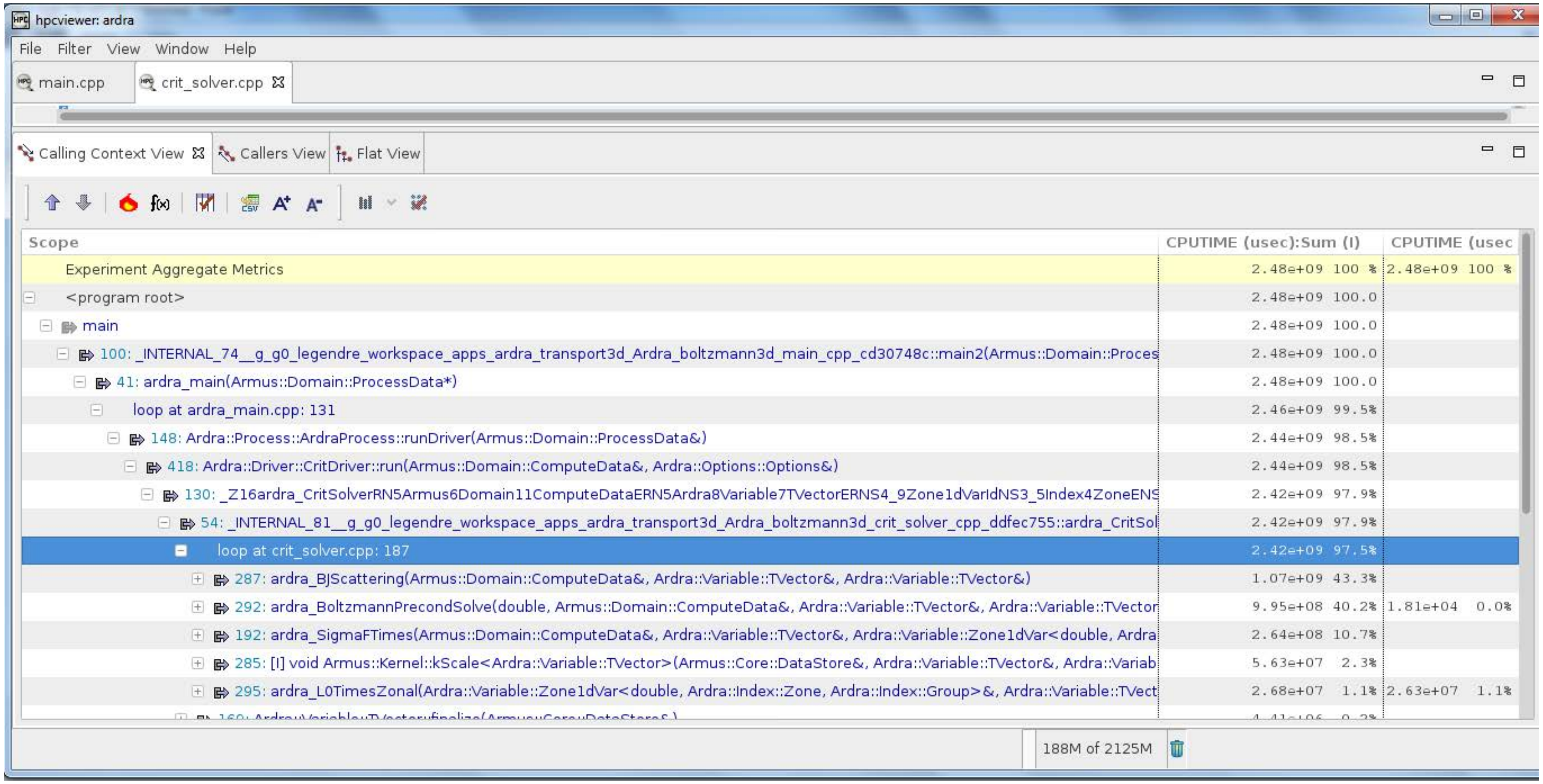
Legend: HPCToolkit, MAP, GPerfTools, Built-In Timers, Open|SpeedShop

Ardra’s Built-In Timers give an aggregated wall-time trace

```
--------------------------------------------------------------------------------------
Run Time Statistics
                                                                    Wall Time (sec)
%Tot %vl            Timer Name                        Count       Total      Per Count
---- ----------------------------------------------- ------------ ---------- ----------

0.99   (0.99)  runDriver                                        1 59.723425  59.723425
0.00     (0.00)  calcDetectorValues                             2  0.000010   0.000005
0.00     (0.00)  fluxEdit                                       1  0.000510   0.000510
0.99     (0.99)  solve                                          1 59.401240  59.401240
0.99       (1.00)  critSolver                                   1 59.401219  59.401219
0.38         (0.38)  ardra_BoltzmannPrecondSolve               25 22.586105   0.903444
0.20           (0.54)  Sweep3D_DD                            1200 12.280318   0.010234
0.02         (0.02)  ardra_L0TimesZonal                        25  0.947570   0.037903
0.63         (0.64)  bj_scattering_fanout                      25 38.082054   1.523282
0.40           (0.63)  bj_gp_loop                            1200 23.807309   0.019839
0.22           (0.35)  bj_lplus                                25 13.394594   0.535784
0.00         (0.00)  dsaMatrixSetup                             1  0.000970   0.000970
0.09         (0.09)  fission_fanout                            26  5.359588   0.206138
0.00     (0.00)  writeRestart                                   1  0.220500   0.220500
```

HPCToolkit can provide CPUTime, Wall-Time, PAPI Counters, MPI Time, Loop-level performance analysis

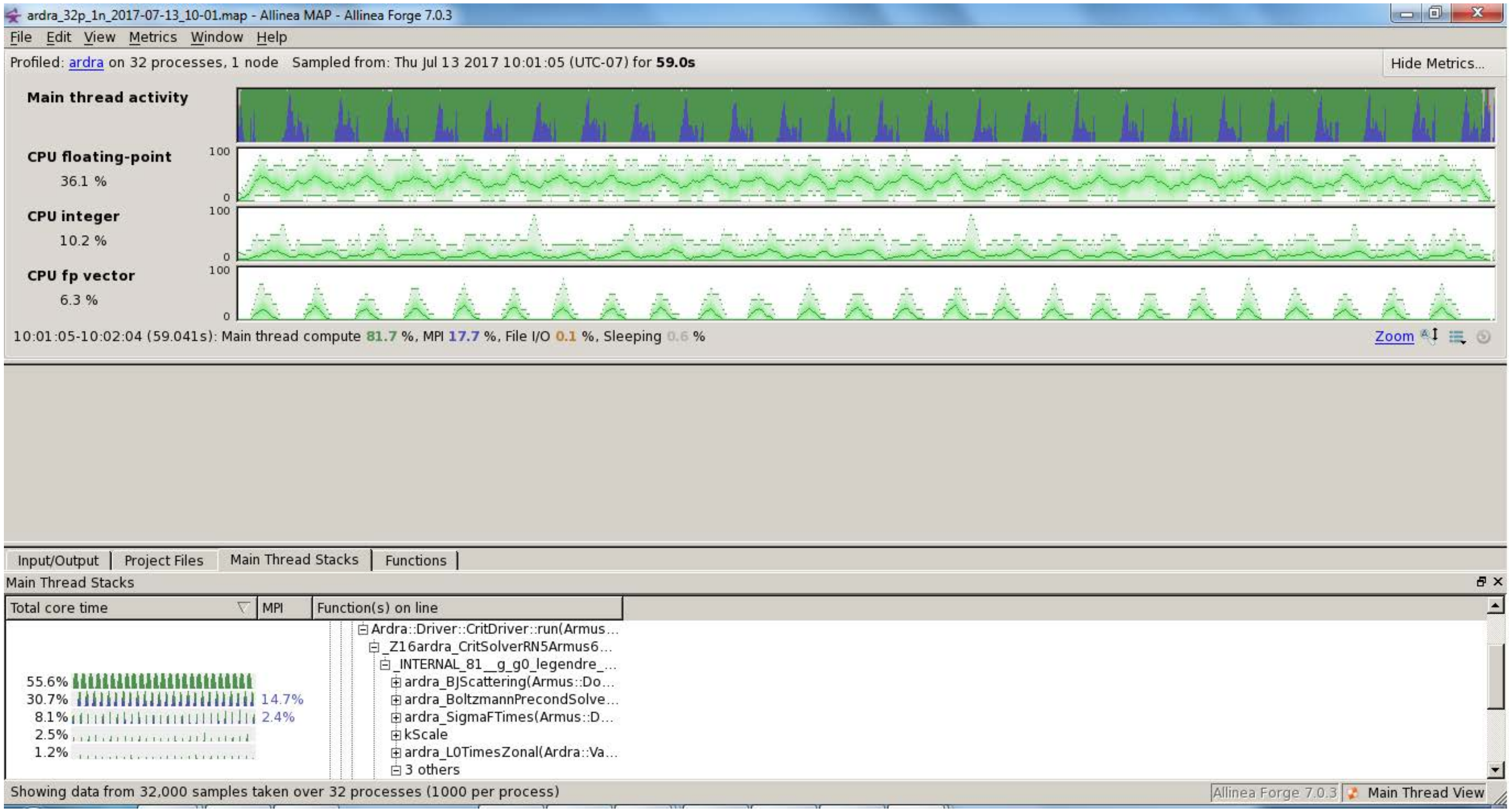


Scope	CPUTIME (usec):Sum (I)	CPUTIME (usec
Experiment Aggregate Metrics	2.48e+09 100 %	2.48e+09 100 %
<program root>	2.48e+09 100.0	
main	2.48e+09 100.0	
100: _INTERNAL_74__g_g0_legendre_workspace_apps_ardra_transport3d_Ardra_boltzmann3d_main_cpp_cd30748c::main2(Armus::Domain::Proces	2.48e+09 100.0	
41: ardra_main(Armus::Domain::ProcessData*)	2.48e+09 100.0	
loop at ardra_main.cpp: 131	2.46e+09 99.5%	
148: Ardra::Process::ArdraProcess::runDriver(Armus::Domain::ProcessData&)	2.44e+09 98.5%	
418: Ardra::Driver::CritDriver::run(Armus::Domain::ComputeData&, Ardra::Options::Options&)	2.44e+09 98.5%	
130: _Z16ardra_CritSolverRN5Armus6Domain11ComputeDataERN5Ardra8Variable7TVectorERNS4_9Zone1dVarIdNS3_5Index4ZoneENS	2.42e+09 97.9%	
54: _INTERNAL_81__g_g0_legendre_workspace_apps_ardra_transport3d_Ardra_boltzmann3d_crit_solver_cpp_ddfec755::ardra_CritSol	2.42e+09 97.9%	
loop at crit_solver.cpp: 187	2.42e+09 97.5%	
287: ardra_BJScattering(Armus::Domain::ComputeData&, Ardra::Variable::TVector&, Ardra::Variable::TVector&)	1.07e+09 43.3%	
292: ardra_BoltzmannPrecondSolve(double, Armus::Domain::ComputeData&, Ardra::Variable::TVector&, Ardra::Variable::TVector	9.95e+08 40.2%	1.81e+04 0.0%
192: ardra_SigmaFTimes(Armus::Domain::ComputeData&, Ardra::Variable::TVector&, Ardra::Variable::Zone1dVar<double, Ardra	2.64e+08 10.7%	
285: [I] void Armus::Kernel::kScale<Ardra::Variable::TVector>(Armus::Core::DataStore&, Ardra::Variable::TVector&, Ardra::Variab	5.63e+07 2.3%	
295: ardra_L0TimesZonal(Ardra::Variable::Zone1dVar<double, Ardra::Index::Zone, Ardra::Index::Group>&, Ardra::Variable::TVect	2.68e+07 1.1%	2.63e+07 1.1%
160: Ardra::Variable::TVector::finalize(Armus::Core::DataStore&)	4.41e+06 0.2%	

Google’s GPerfTools provides an easy-to-use time breakdown

```
Total: 5536
1409  25.5%  25.5%     1410  25.5% ardra_LTimes::{lambda#2}::operator (inline)
 993  17.9%  43.4%     1000  18.1% ardra_LPlusTimes::{lambda#1}::operator (inline)
 855  15.4%  58.8%      855  15.4% Ardra::Sweep_Solver::Sweep_Solver3D::SweepDD
 483   8.7%  67.6%      613  11.1% ::applySigS
 477   8.6%  76.2%     3198  57.8% RAJA::forall (inline)
 324   5.9%  82.0%      324   5.9% Armus::Kernel::kGenericElemental2
 101   1.8%  83.9%      101   1.8% __intel_avx_rep_memset
  98   1.8%  85.6%      113   2.0% RAJA::LayoutBase_impl::operator (inline)
  85   1.5%  87.2%       85   1.5% PMPI_Testany
```

Allinea MAP Provides Wall Time, MPI Time, CPU Time, instruction mixes, FP vectorization rates



ardra_32p_1n_2017-07-13_10-01.map - Allinea MAP - Allinea Forge 7.0.3

Profiled: ardra on 32 processes, 1 node   Sampled from: Thu Jul 13 2017 10:01:05 (UTC-07) for **59.0s**

Main thread activity

CPU floating-point 36.1 %

CPU integer 10.2 %

CPU fp vector 6.3 %

10:01:05-10:02:04 (59.041s): Main thread compute 81.7 %, MPI 17.7 %, File I/O 0.1 %, Sleeping 0.6 %

Main Thread Stacks

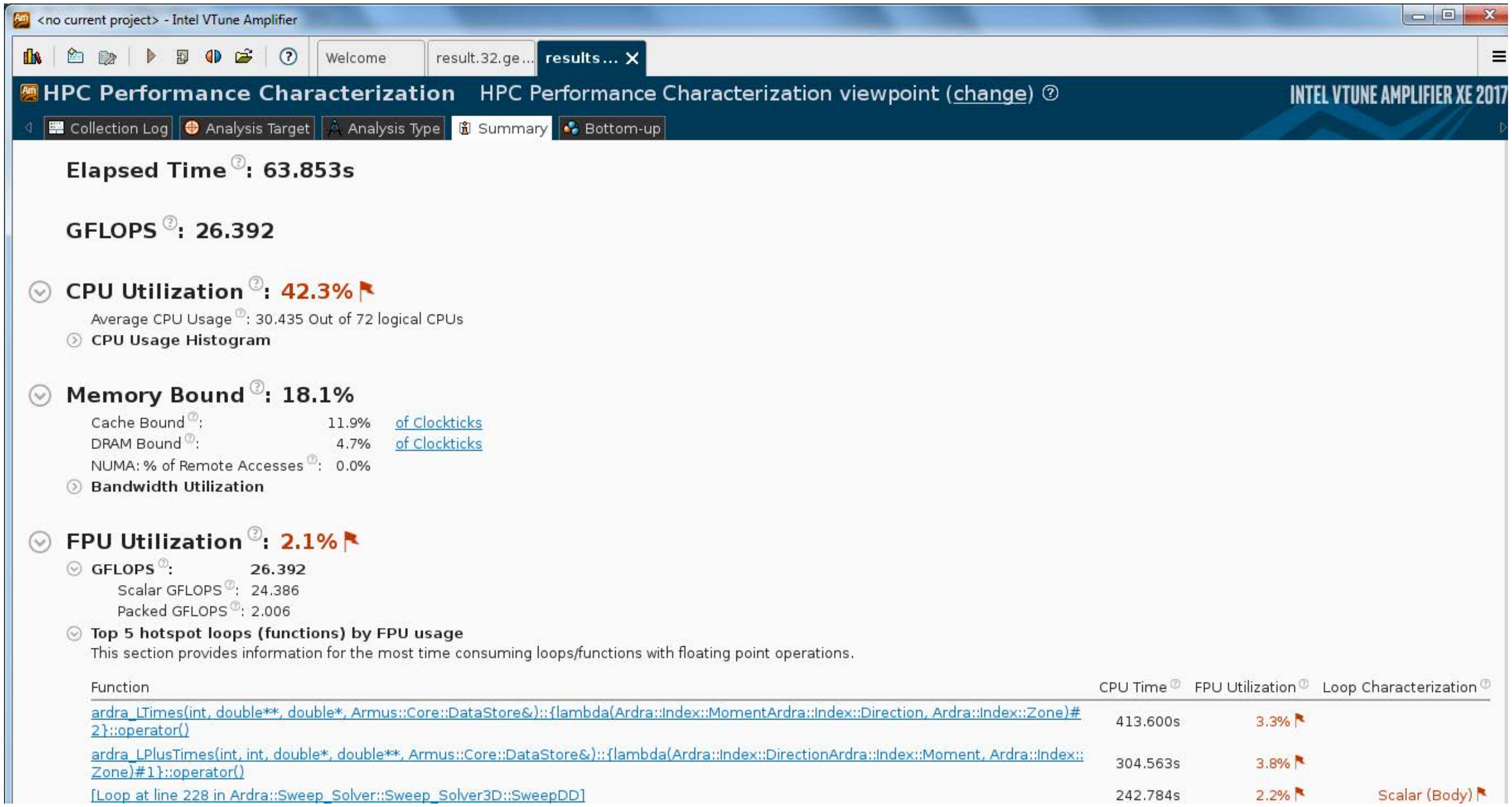
Total core time	MPI	Function(s) on line
		⊟ Ardra::Driver::CritDriver::run(Armus…
		⊟ _Z16ardra_CritSolverRN5Armus6…
		⊟ _INTERNAL_81__g_g0_legendre_…
55.6%		⊞ ardra_BJScattering(Armus::Do…
30.7%	14.7%	⊞ ardra_BoltzmannPrecondSolve…
8.1%	2.4%	⊞ ardra_SigmaFTimes(Armus::D…
2.5%		⊞ kScale
1.2%		⊞ ardra_L0TimesZonal(Ardra::Va…
		⊟ 3 others

Showing data from 32,000 samples taken over 32 processes (1000 per process)

VTune and Vectorizing Ardra

- VTune Suggested we should vectorize the LTimes and LPlusTimes kernels (see next few slides)
- Vectorizing LTimes was a significant effort.
- Vectorization significantly speeds up Ardra for non memory-bound problem sizes.

Intel VTune Suggests FPU Utilization Issues across Ardra


<no current project> - Intel VTune Amplifier
Welcome
result.32.ge...
results...
HPC Performance Characterization
HPC Performance Characterization viewpoint (change)
INTEL VTUNE AMPLIFIER XE 2017
Collection Log
Analysis Target
Analysis Type
Summary
Bottom-up

Elapsed Time: 63.853s

GFLOPS: 26.392

CPU Utilization: 42.3%

Average CPU Usage: 30.435 Out of 72 logical CPUs

CPU Usage Histogram

Memory Bound: 18.1%

Cache Bound:	11.9%	of Clockticks
DRAM Bound:	4.7%	of Clockticks
NUMA: % of Remote Accesses:	0.0%	

Bandwidth Utilization

FPU Utilization: 2.1%

GFLOPS: 26.392

Scalar GFLOPS: 24.386

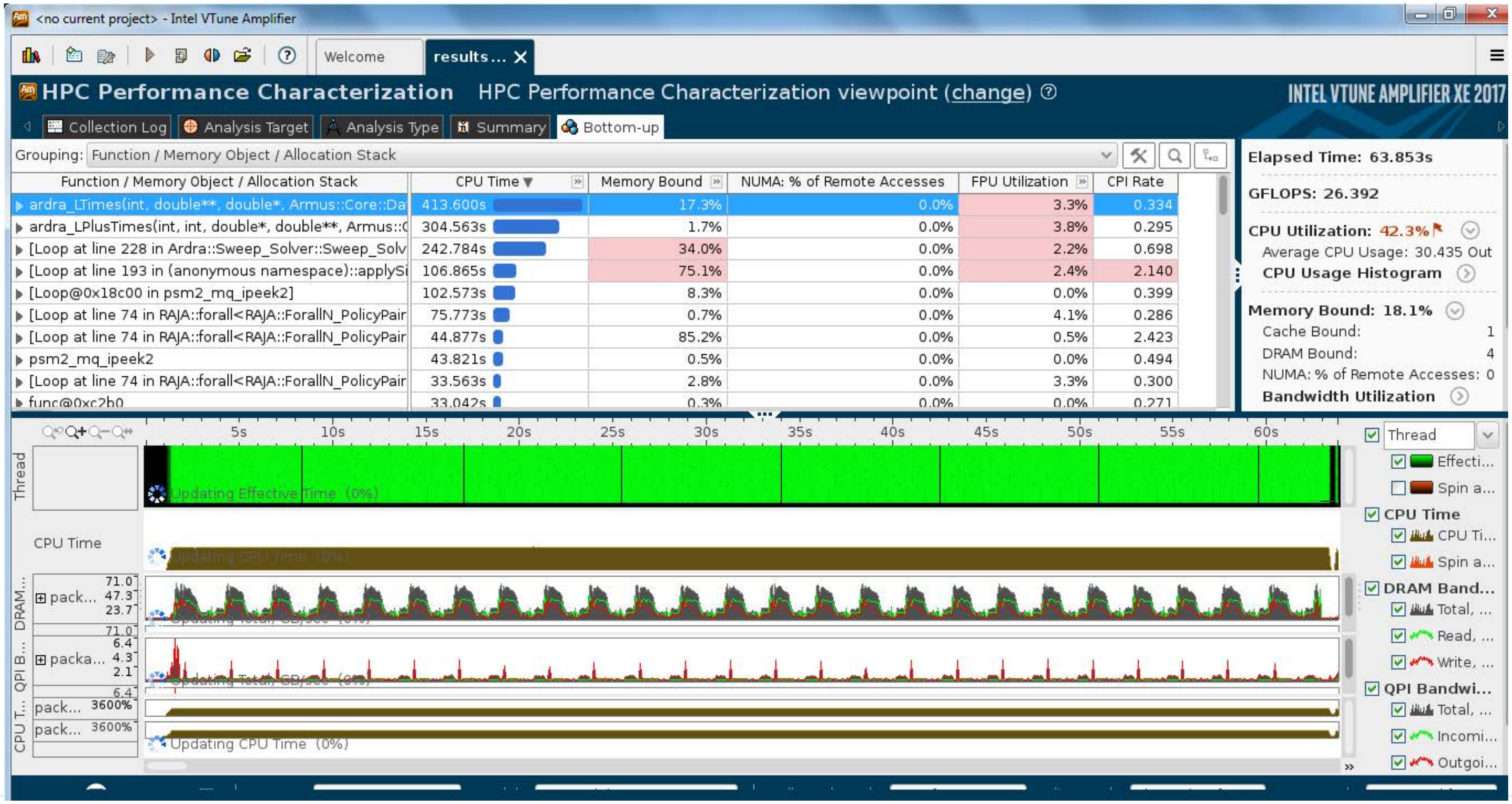
Packed GFLOPS: 2.006

Top 5 hotspot loops (functions) by FPU usage

This section provides information for the most time consuming loops/functions with floating point operations.

Function	CPU Time	FPU Utilization	Loop Characterization
ardra_LTimes(int, double**, double*, Armus::Core::DataStore&)::{lambda(Ardra::Index::MomentArdra::Index::Direction, Ardra::Index::Zone)#2}::operator()	413.600s	3.3%	
ardra_LPlusTimes(int, int, double*, double**, Armus::Core::DataStore&)::{lambda(Ardra::Index::DirectionArdra::Index::Moment, Ardra::Index::Zone)#1}::operator()	304.563s	3.8%	
[Loop at line 228 in Ardra::Sweep_Solver::Sweep_Solver3D::SweepDD]	242.784s	2.2%	Scalar (Body)

Intel VTune Suggests FPU Utilization Issues in LTimes and LPlusTimes kernels



<no current project> - Intel VTune Amplifier

Welcome | results…

HPC Performance Characterization — HPC Performance Characterization viewpoint (change)

INTEL VTUNE AMPLIFIER XE 2017

Collection Log | Analysis Target | Analysis Type | Summary | Bottom-up

Grouping: Function / Memory Object / Allocation Stack

Function / Memory Object / Allocation Stack	CPU Time ▼	Memory Bound	NUMA: % of Remote Accesses	FPU Utilization	CPI Rate
ardra_LTimes(int, double**, double*, Armus::Core::Da	413.600s	17.3%	0.0%	3.3%	0.334
ardra_LPlusTimes(int, int, double*, double**, Armus::(	304.563s	1.7%	0.0%	3.8%	0.295
[Loop at line 228 in Ardra::Sweep_Solver::Sweep_Solv	242.784s	34.0%	0.0%	2.2%	0.698
[Loop at line 193 in (anonymous namespace)::applySi	106.865s	75.1%	0.0%	2.4%	2.140
[Loop@0x18c00 in psm2_mq_ipeek2]	102.573s	8.3%	0.0%	0.0%	0.399
[Loop at line 74 in RAJA::forall<RAJA::ForallN_PolicyPair	75.773s	0.7%	0.0%	4.1%	0.286
[Loop at line 74 in RAJA::forall<RAJA::ForallN_PolicyPair	44.877s	85.2%	0.0%	0.5%	2.423
psm2_mq_ipeek2	43.821s	0.5%	0.0%	0.0%	0.494
[Loop at line 74 in RAJA::forall<RAJA::ForallN_PolicyPair	33.563s	2.8%	0.0%	3.3%	0.300
func@0xc2b0	33.042s	0.3%	0.0%	0.0%	0.271

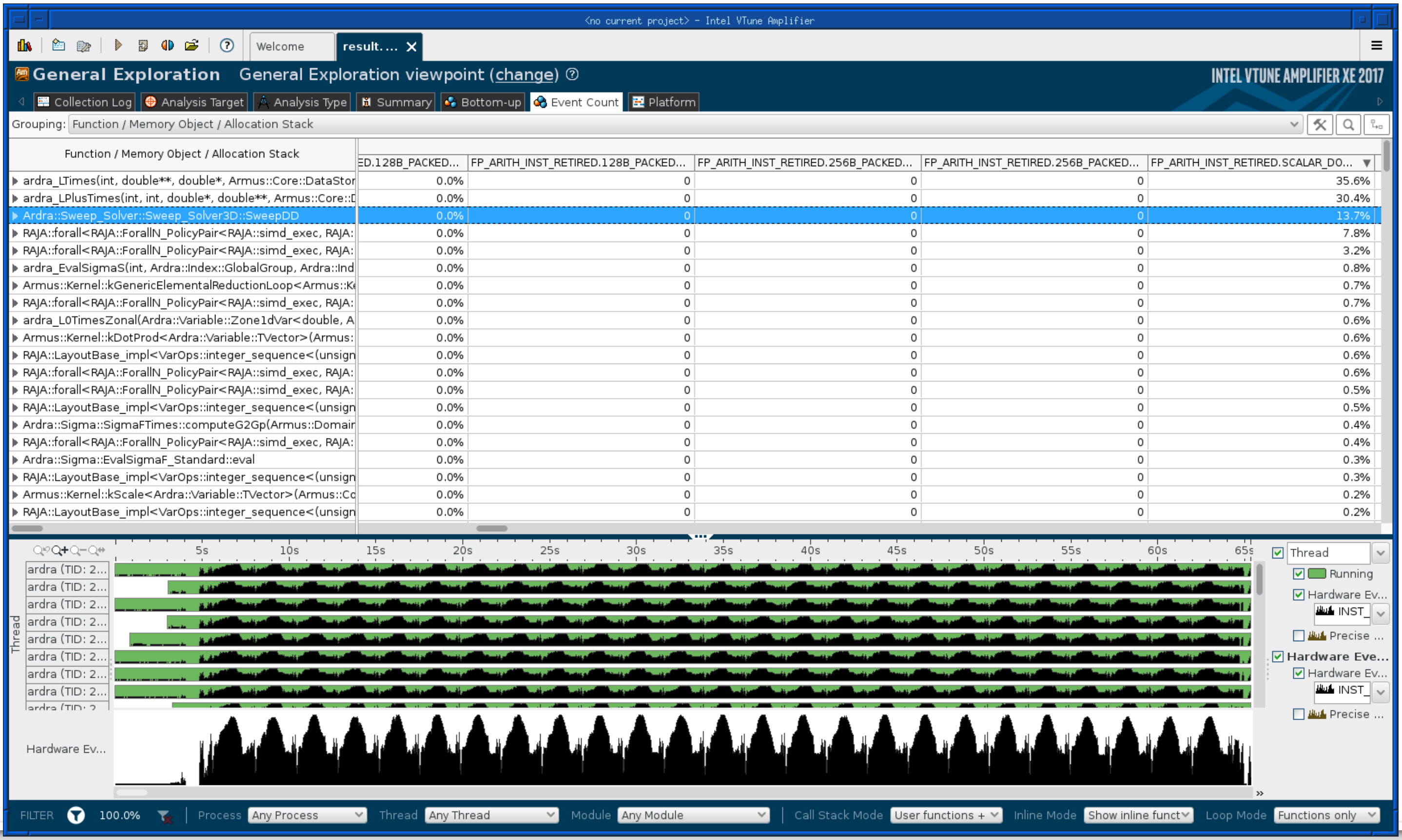
Elapsed Time: 63.853s

GFLOPS: 26.392

CPU Utilization: 42.3%
Average CPU Usage: 30.435 Out
CPU Usage Histogram

Memory Bound: 18.1%
Cache Bound: 1
DRAM Bound: 4
NUMA: % of Remote Accesses: 0
Bandwidth Utilization

VTune Shows No Vectorization Happening in Ardra

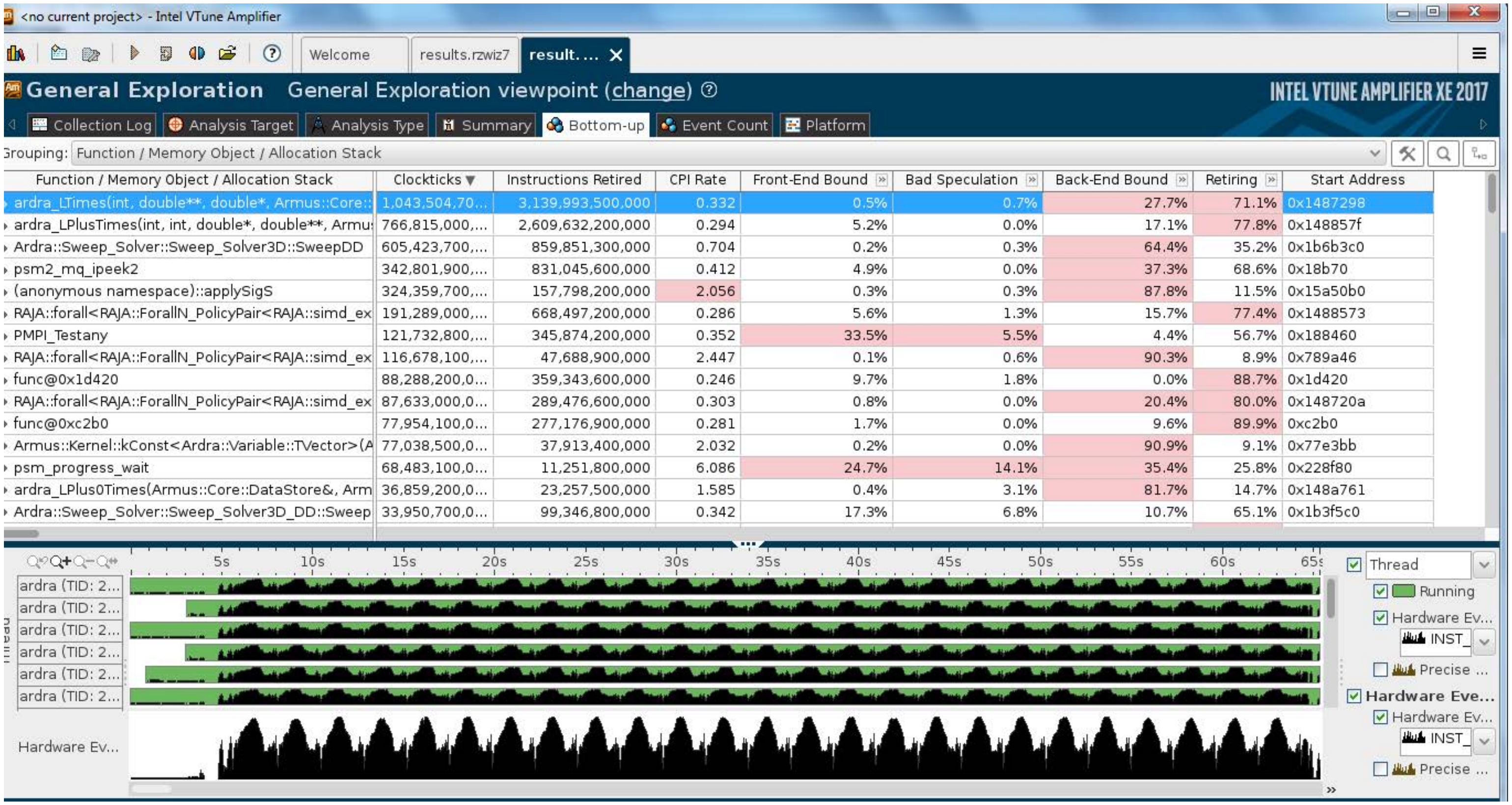


Grouping: Function / Memory Object / Allocation Stack

Function / Memory Object / Allocation Stack	ED.128B_PACKED...	FP_ARITH_INST_RETIRED.128B_PACKED...	FP_ARITH_INST_RETIRED.256B_PACKED...	FP_ARITH_INST_RETIRED.256B_PACKED...	FP_ARITH_INST_RETIRED.SCALAR_DO... ▼
▸ ardra_LTimes(int, double**, double*, Armus::Core::DataStor	0.0%	0	0	0	35.6%
▸ ardra_LPlusTimes(int, int, double*, double**, Armus::Core::D	0.0%	0	0	0	30.4%
▸ Ardra::Sweep_Solver::Sweep_Solver3D::SweepDD	0.0%	0	0	0	13.7%
▸ RAJA::forall<RAJA::ForallN_PolicyPair<RAJA::simd_exec, RAJA:	0.0%	0	0	0	7.8%
▸ RAJA::forall<RAJA::ForallN_PolicyPair<RAJA::simd_exec, RAJA:	0.0%	0	0	0	3.2%
▸ ardra_EvalSigmaS(int, Ardra::Index::GlobalGroup, Ardra::Ind	0.0%	0	0	0	0.8%
▸ Armus::Kernel::kGenericElementalReductionLoop<Armus::Ke	0.0%	0	0	0	0.7%
▸ RAJA::forall<RAJA::ForallN_PolicyPair<RAJA::simd_exec, RAJA:	0.0%	0	0	0	0.7%
▸ ardra_L0TimesZonal(Ardra::Variable::Zone1dVar<double, A	0.0%	0	0	0	0.6%
▸ Armus::Kernel::kDotProd<Ardra::Variable::TVector>(Armus:	0.0%	0	0	0	0.6%
▸ RAJA::LayoutBase_impl<VarOps::integer_sequence<(unsign	0.0%	0	0	0	0.6%
▸ RAJA::forall<RAJA::ForallN_PolicyPair<RAJA::simd_exec, RAJA:	0.0%	0	0	0	0.6%
▸ RAJA::forall<RAJA::ForallN_PolicyPair<RAJA::simd_exec, RAJA:	0.0%	0	0	0	0.5%
▸ RAJA::LayoutBase_impl<VarOps::integer_sequence<(unsign	0.0%	0	0	0	0.5%
▸ Ardra::Sigma::SigmaFTimes::computeG2Gp(Armus::Domair	0.0%	0	0	0	0.4%
▸ RAJA::forall<RAJA::ForallN_PolicyPair<RAJA::simd_exec, RAJA:	0.0%	0	0	0	0.4%
▸ Ardra::Sigma::EvalSigmaF_Standard::eval	0.0%	0	0	0	0.3%
▸ RAJA::LayoutBase_impl<VarOps::integer_sequence<(unsign	0.0%	0	0	0	0.3%
▸ Armus::Kernel::kScale<Ardra::Variable::TVector>(Armus::Cc	0.0%	0	0	0	0.2%
▸ RAJA::LayoutBase_impl<VarOps::integer_sequence<(unsign	0.0%	0	0	0	0.2%

FILTER 100.0% | Process: Any Process | Thread: Any Thread | Module: Any Module | Call Stack Mode: User functions + | Inline Mode: Show inline funct | Loop Mode: Functions only

VTune Shows Instruction Retirement Issues in LTimes and LPlusTimes



Grouping: Function / Memory Object / Allocation Stack

Function / Memory Object / Allocation Stack	Clockticks ▼	Instructions Retired	CPI Rate	Front-End Bound	Bad Speculation	Back-End Bound	Retiring	Start Address
ardra_LTimes(int, double**, double*, Armus::Core::	1,043,504,70...	3,139,993,500,000	0.332	0.5%	0.7%	27.7%	71.1%	0x1487298
ardra_LPlusTimes(int, int, double*, double**, Armu:	766,815,000,...	2,609,632,200,000	0.294	5.2%	0.0%	17.1%	77.8%	0x148857f
Ardra::Sweep_Solver::Sweep_Solver3D::SweepDD	605,423,700,...	859,851,300,000	0.704	0.2%	0.3%	64.4%	35.2%	0x1b6b3c0
psm2_mq_ipeek2	342,801,900,...	831,045,600,000	0.412	4.9%	0.0%	37.3%	68.6%	0x18b70
(anonymous namespace)::applySigS	324,359,700,...	157,798,200,000	2.056	0.3%	0.3%	87.8%	11.5%	0x15a50b0
RAJA::forall<RAJA::ForallN_PolicyPair<RAJA::simd_ex	191,289,000,...	668,497,200,000	0.286	5.6%	1.3%	15.7%	77.4%	0x1488573
PMPI_Testany	121,732,800,...	345,874,200,000	0.352	33.5%	5.5%	4.4%	56.7%	0x188460
RAJA::forall<RAJA::ForallN_PolicyPair<RAJA::simd_ex	116,678,100,...	47,688,900,000	2.447	0.1%	0.6%	90.3%	8.9%	0x789a46
func@0x1d420	88,288,200,0...	359,343,600,000	0.246	9.7%	1.8%	0.0%	88.7%	0x1d420
RAJA::forall<RAJA::ForallN_PolicyPair<RAJA::simd_ex	87,633,000,0...	289,476,600,000	0.303	0.8%	0.0%	20.4%	80.0%	0x148720a
func@0xc2b0	77,954,100,0...	277,176,900,000	0.281	1.7%	0.0%	9.6%	89.9%	0xc2b0
Armus::Kernel::kConst<Ardra::Variable::TVector>(A	77,038,500,0...	37,913,400,000	2.032	0.2%	0.0%	90.9%	9.1%	0x77e3bb
psm_progress_wait	68,483,100,0...	11,251,800,000	6.086	24.7%	14.1%	35.4%	25.8%	0x228f80
ardra_LPlus0Times(Armus::Core::DataStore&, Arm	36,859,200,0...	23,257,500,000	1.585	0.4%	3.1%	81.7%	14.7%	0x148a761
Ardra::Sweep_Solver::Sweep_Solver3D_DD::Sweep	33,950,700,0...	99,346,800,000	0.342	17.3%	6.8%	10.7%	65.1%	0x1b3f5c0

Intel Optimization Manual Says High Retirement Means You Should Vectorize

Intel 64 and IA-32 Architectures Optimization Reference Manual section B.1.7:

A high Retiring value for non-vectorized code may be a good hint to vectorize the code. Doing so essentially lets more operations to be [SIC] completed by single instruction/micro-op; hence improve performance.

VTune Vectorization Summary

- **Hypothesis**: We should vectorize LTimes and LPlusTimes:

- VTune Provided Evidence:
  - Low FPU Utilization in LTimes and LPlusTimes
  - No Vector instructions being compiled into LTimes and LPlusTimes
    - Also backed by MAP and disassembling the kernels
  - LTimes and LPlusTimes spends most time retiring instructions
  - Intel optimization manual says retiring is optimized via vectorization

- Used Intel compiler intrinsics to vectorize LTimes into a loop around a 256-bit fused add/multiply instruction.

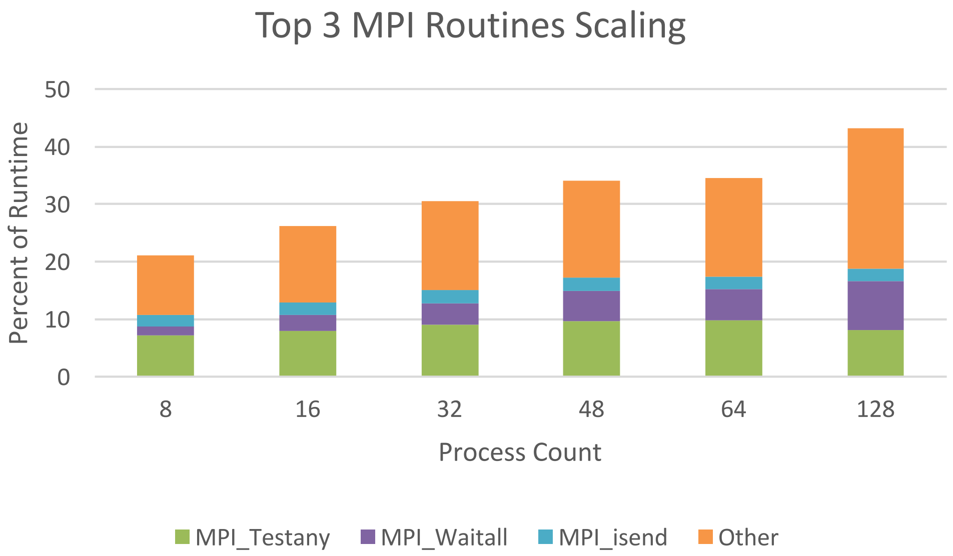
- **Results:** Vectorizing LTimes sped up by over 2x. Problem now runs at the speed of L2 cache

MPI Weak Scaling with Ardra

- The sweep algorithms used by Ardra are known not to scale
- Ardra has previously been shown to scale as well as the ideal sweep algorithm
- The mpiP tool can break down MPI times by call path and point-to-point vs. collective

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory
under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Scaling Top 3 MPI Routines in Ardra



Other Measurements

- Memory High Water Mark
- Memory Bandwidth
- IPC

memP Tool Shows Memory High-Water Mark

```
-------------------------------------------------------------------------------
@--- Greatest Heap High Water Mark (top 5, descending, bytes)
-------------------------------------------------------------------------------
      Rank      Heap HWM       Stack          Sum
        16     334.45 MB    27.55 KB    334.48 MB
         1     334.27 MB    27.64 KB    334.30 MB
         4     334.27 MB    27.64 KB    334.30 MB
        17     334.19 MB    27.55 KB    334.21 MB
        20     334.19 MB    27.55 KB    334.21 MB
-------------------------------------------------------------------------------
@--- Heap HWM Statistics -------------------------------------------
-------------------------------------------------------------------------------
Max                               :    334.45 MB
Median                            :    310.12 MB
Mean                              :    310.61 MB
Min                               :    287.77 MB
Stddev                            :     16.32 MB
Coefficient of variation          :     0.052555
```

Performance Co-Pilot Provides Access to Memory Bandwidth Counters

```
Run Time Statistics                                         Wall Time (sec)
     Name                                           Count   Total     Per Count  Read BW (MB/S)  Write BW (MB/S)
---- ------------------------------- -------- ----------- ---------- ---------- ----------------- ---------------

0.99   (0.99)  runDriver                                1  58.194303  58.194303        33521.11        19988.36
0.00     (0.00)  calcDetectorValues                     2   0.001631   0.000816        58550.63         5699.95
0.00     (0.00)  fluxEdit                               1   0.001138   0.001138        19093.88          449.64
0.98     (0.99)  solve                                  1  57.421936  57.421936        33789.96        20242.19
0.98       (1.00)  critSolver                           1  57.421919  57.421919        33787.91        20241.49
0.38         (0.39)  ardra_BoltzmannPrecondSolve       25  22.263067   0.890523        11457.50         6998.44
0.20           (0.53)  Sweep3D_DD                    1200  11.754593   0.009795        20016.79        12478.46
0.01         (0.01)  ardra_L0TimesZonal                25   0.839731   0.033589       107391.46         1972.86
0.65         (0.66)  bj_scattering_fanout              25  38.104032   1.524161        33744.05        18284.05
0.39           (0.60)  bj_gp_loop                    1200  23.008618   0.019174        45400.02        24469.75
0.22           (0.34)  bj_lplus                        25  12.917722   0.516709        12885.62         4581.34
0.00         (0.00)  dsaMatrixSetup                     1   0.000816   0.000816         8204.87         5446.62
0.08         (0.08)  fission_fanout                    26   4.458054   0.171464        47140.92        45936.05
0.01     (0.01)  writeRestart                           1   0.668223   0.668223          412.27          341.27

--------------------------------------------------------------------------------------
```

IPC calculations from PAPI

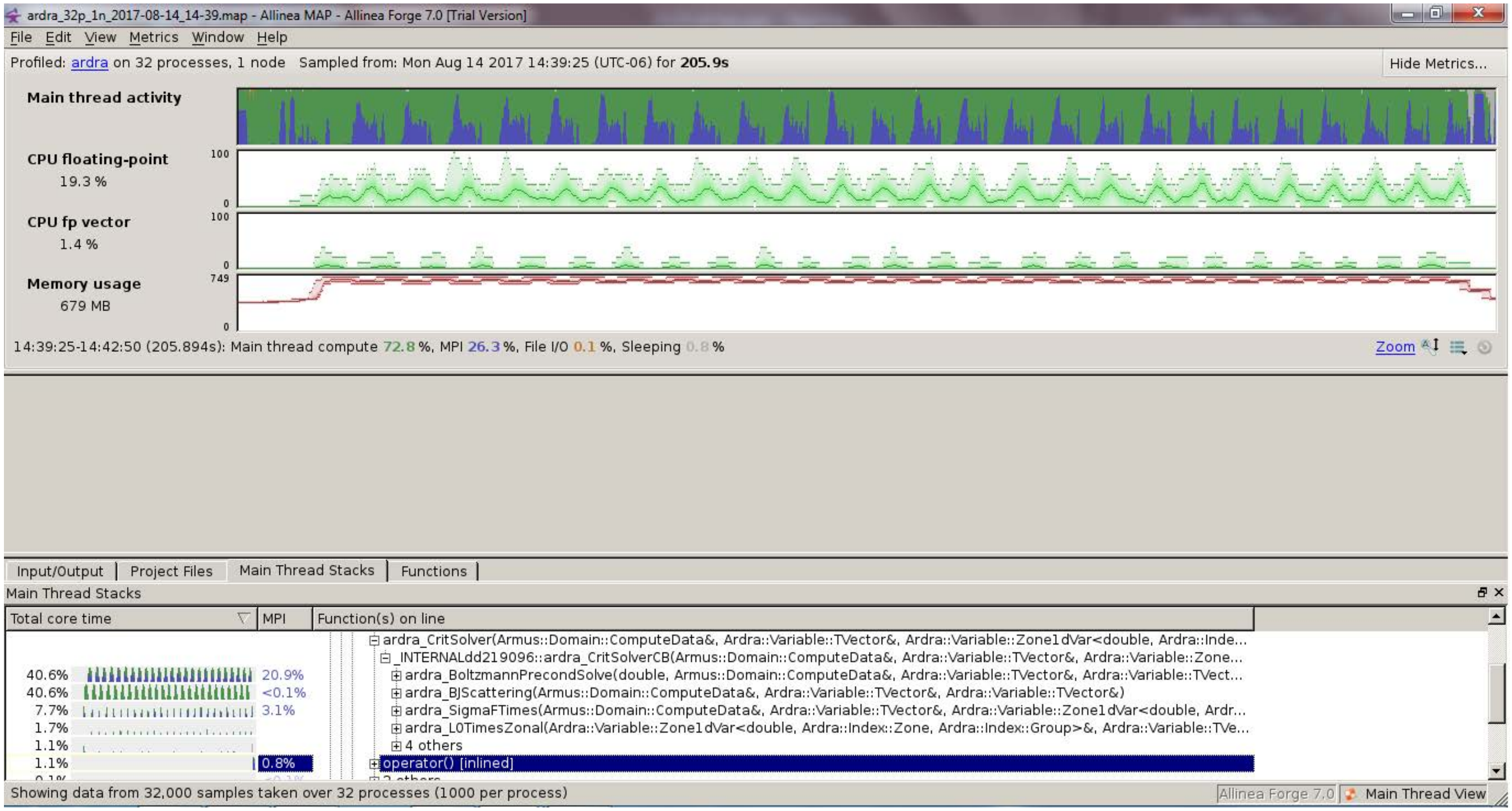
```
                                                   Wall Time (sec)
%Tot    %lvl   Timer Name             Count      Total  Per Count    PAPI_TOT_INS    PAPI_TOT_CYC     IPC
---- --------- ---------------------------- ----------- ---------- -------------- ---------------    ----
0.99   (0.99)  runDriver                  1  55.470365  55.470365    324162081674    137489604904    2.36
0.00     (0.00)  calcDetectorValues       2   0.000031   0.000015            4766           11925    0.40
0.00     (0.00)  fluxEdit                 1   0.000435   0.000435         1416821         1026943    1.38
0.98     (0.99)  solve                    1  54.835454  54.835454    323421631027    137001207761    2.36
0.98       (1.00)  critSolver             1  54.835402  54.835402    323421624234    137001189017    2.36
0.35         (0.35)  ardra_BoltzmannPrecondSolve 25 19.416221 0.776649 462818928     28673595986    1.61
0.20           (0.58)  Sweep3D_DD      1200  11.347921   0.009457     41570458833     26857361098    1.55
0.01         (0.01)  ardra_L0TimesZonal 25    0.817820   0.032713      6106458767      2090150611    2.92
0.64         (0.66)  bj_scattering_fanout 25  35.924563 1.436983    245742185602     90729230958    2.71
0.40           (0.62)  bj_gp_loop      1200  22.161912   0.018468    133669003312     56761088769    2.35
0.23           (0.35)  bj_lplus          25  12.657956   0.506318    111588764852     32601102609    3.42
0.00         (0.00)  dsaMatrixSetup       1   0.000040   0.000040            2511            9463    0.27
0.08         (0.08)  fission_fanout      26   4.409059   0.169579     22592708167     10981676930    2.06
0.01     (0.01)  writeRestart             1   0.534155   0.534155       358056618       236388890    1.51
```

Lawrence Livermore
National Laboratory

KNL Data for Ardra

- Run on LANL’s Trinitite

Allinea MAP on KNL Showing Instruction Mix



ardra_32p_1n_2017-08-14_14-39.map - Allinea MAP - Allinea Forge 7.0 [Trial Version]

Profiled: ardra on 32 processes, 1 node   Sampled from: Mon Aug 14 2017 14:39:25 (UTC-06) for 205.9s

Main thread activity

CPU floating-point
19.3 %

CPU fp vector
1.4 %

Memory usage
679 MB

14:39:25-14:42:50 (205.894s): Main thread compute 72.8 %, MPI 26.3 %, File I/O 0.1 %, Sleeping 0.8 %

Main Thread Stacks

```
Total core time   MPI     Function(s) on line
                          ⊟ardra_CritSolver(Armus::Domain::ComputeData&, Ardra::Variable::TVector&, Ardra::Variable::Zone1dVar<double, Ardra::Inde...
                           ⊟_INTERNALdd219096::ardra_CritSolverCB(Armus::Domain::ComputeData&, Ardra::Variable::TVector&, Ardra::Variable::Zone...
40.6%             20.9%      ⊞ardra_BoltzmannPrecondSolve(double, Armus::Domain::ComputeData&, Ardra::Variable::TVector&, Ardra::Variable::TVect...
40.6%             <0.1%      ⊞ardra_BJScattering(Armus::Domain::ComputeData&, Ardra::Variable::TVector&, Ardra::Variable::TVector&)
 7.7%             3.1%       ⊞ardra_SigmaFTimes(Armus::Domain::ComputeData&, Ardra::Variable::TVector&, Ardra::Variable::Zone1dVar<double, Ardr...
 1.7%                        ⊞ardra_L0TimesZonal(Ardra::Variable::Zone1dVar<double, Ardra::Index::Zone, Ardra::Index::Group>&, Ardra::Variable::TVe...
 1.1%                        ⊞4 others
 1.1%             0.8%    ⊞operator() [inlined]
 0.1%             <0.1%   ⊞2 others
```

Showing data from 32,000 samples taken over 32 processes (1000 per process)

KNL CrayPAT Hardware Counters

```
Table 4:  Program HW Performance Counter Data

PE=HIDE


==============================================================================
  Total
------------------------------------------------------------------------------
  UNHALTED_CORE_CYCLES             148,513,518,999
  UNHALTED_REFERENCE_CYCLES        138,612,610,990
  INSTRUCTION_RETIRED              121,483,923,581
  LLC_REFERENCES                     2,923,544,676
  LLC_MISSES                            91,517,871
  LLC cache hit,miss ratio  96.9% hits        3.1% misses
==============================================================================
```

Ardra’s Built-In Timers, with HBM

```
--------------------------------------------------------------------------------------
Run Time Statistics
                                                                    Wall Time (sec)
%Tot %vl            Timer Name                        Count       Total     Per Count
---- ------------------------------------------------ ----------- ---------- ----------
0.86   (0.86)  runDriver                                        1 101.049208 101.049208
0.00     (0.00)  calcDetectorValues                             2   0.000017   0.000009
0.00     (0.00)  fluxEdit                                       1   0.014949   0.014949
0.84     (0.98)  solve                                          1  99.364175  99.364175
0.84       (1.00)  critSolver                                   1  99.364004  99.364004
0.45         (0.54)  ardra_BoltzmannPrecondSolve               25  53.269412   2.130776
0.27           (0.60)  Sweep3D_DD                            1200  31.825563   0.026521
0.02         (0.02)  ardra_L0TimesZonal                        25   2.202952   0.088118
0.44         (0.52)  bj_scattering_fanout                      25  51.718966   2.068759
0.30           (0.67)  bj_gp_loop                            1200  34.733989   0.028945
0.14           (0.31)  bj_lplus                                25  16.131730   0.645269
0.00         (0.00)  dsaMatrixSetup                             1   0.000052   0.000052
0.08         (0.09)  fission_fanout                            26   9.390708   0.361181
0.01     (0.02)  writeRestart                                   1   1.615085   1.615085
```

Ardra’s Built-In Timers, with HBM disabled

```
--------------------------------------------------------------------------------------
Run Time Statistics
                                                                    Wall Time (sec)
%Tot %vl            Timer Name                        Count       Total      Per Count
---- ----------------------------------------------- ----------- ---------- ----------
0.88   (0.88)  runDriver                                       1 117.428259 117.428259
0.00     (0.00)  calcDetectorValues                            2   0.000019   0.000010
0.00     (0.00)  fluxEdit                                      1   0.012143   0.012143
0.87     (0.99)  solve                                         1 115.775249 115.775249
0.87       (1.00)  critSolver                                  1 115.774777 115.774777
0.39         (0.45)  ardra_BoltzmannPrecondSolve              25  52.312439   2.092498
0.23           (0.59)  Sweep3D_DD                           1200  31.040346   0.025867
0.02         (0.02)  ardra_L0TimesZonal                       25   2.198163   0.087927
0.50         (0.57)  bj_scattering_fanout                     25  66.402436   2.656097
0.36           (0.73)  bj_gp_loop                           1200  48.644699   0.040537
0.12           (0.24)  bj_lplus                               25  16.083930   0.643357
0.00         (0.00)  dsaMatrixSetup                            1   0.000026   0.000026
0.08         (0.09)  fission_fanout                           26  10.131977   0.389691
0.01     (0.01)  writeRestart                                  1   1.543792   1.543792
```

Ardra’s Built-In Timers with IPC from PAPI

```
--------------------------------------------------------------------------------------
Run Time Statistics
                                                          Wall Time (sec)
%Tot %vl            Timer Name                   Count    Total    Per Count    PAPI_TOT_INS     PAPI_TOT_CYC    IPC
---- -------------------------------------------- ---- ---------- ---------- --------------- --------------- ------
0.85   (0.85)  runDriver                             1  98.596496  98.596496    127939335250    136701452657   0.94
0.00     (0.00)  calcDetectorValues                  2   0.000068   0.000034            4394           26922   0.16
0.00     (0.00)  fluxEdit                            1   0.011763   0.011763          764245         1192409   0.64
0.84     (0.98)  solve                               1  97.051414  97.051414    127129798718    134946610048   0.94
0.84       (1.00)  critSolver                        1  97.051194  97.051194    127129792939    134946537515   0.94
0.44         (0.52)  ardra_BoltzmannPrecondSolve    25  50.812436   2.032497     34422457161     55545714728   0.62
0.27           (0.62)  Sweep3D_DD                 1200  31.608640   0.026341     29440200696     46646236415   0.63
0.02         (0.02)  ardra_L0TimesZonal             25   2.175385   0.087015      3168726418      3177950528   1.00
0.43         (0.52)  bj_scattering_fanout           25  50.008220   2.000329     77215472964     62496507175   1.24
0.29           (0.67)  bj_gp_loop                 1200  33.539460   0.027950     46714375423     41527300851   1.12
0.13           (0.31)  bj_lplus                     25  15.503903   0.620156     30440374680     20618413930   1.48
0.00         (0.00)  dsaMatrixSetup                  1   0.000083   0.000083            2567           20830   0.12
0.08         (0.09)  fission_fanout                 26   9.211234   0.354278     10944412869     12622144189   0.87
0.01     (0.02)  writeRestart                        1   1.536780   1.536780       617062571      1529302924   0.40
```

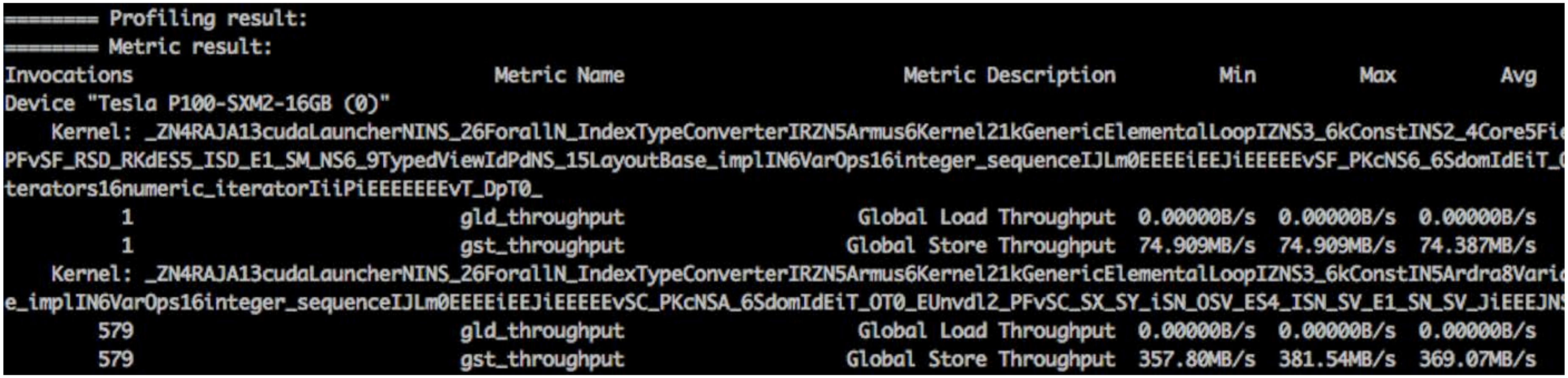
Using the LTimes Vector Optimizations from CTS-1 has smaller impact on KNL

- LTimes when scalar: 31.68s
- LTimes after vector optimizations: 34.88s

GPU Raw Data

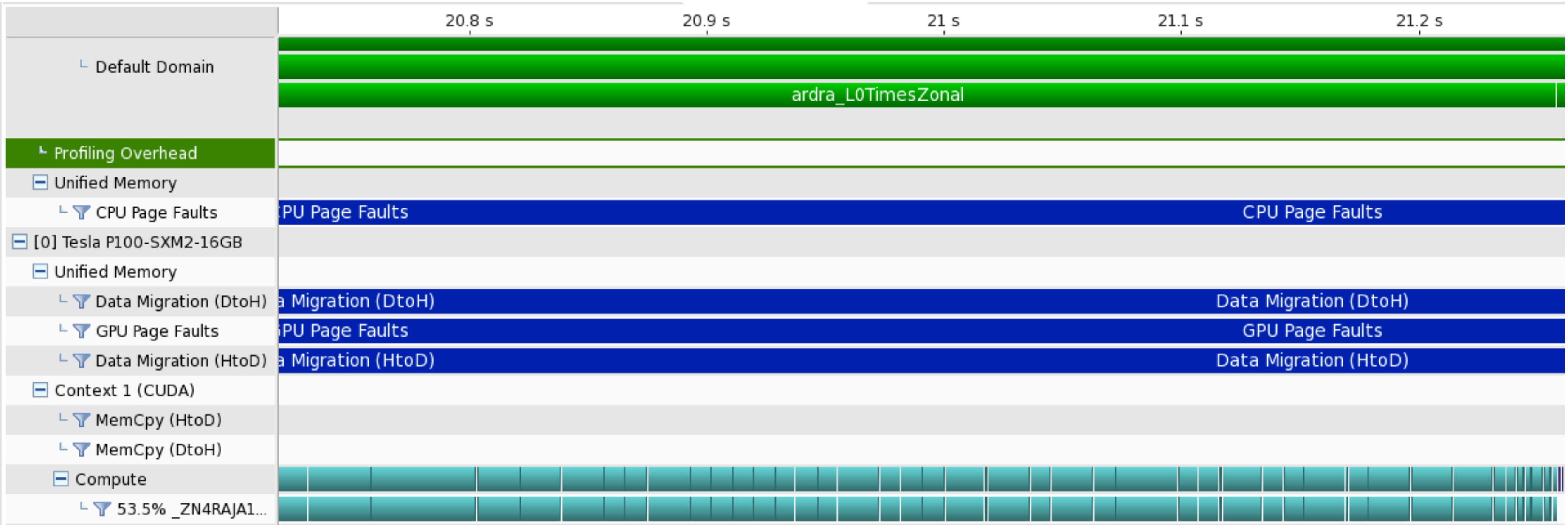
- These benchmarks were run on rzmanta, a Coral EA system at LLNL. Some key specs
  - Each node has 4 Tesla P100-SXM2 Nvidia GPUs
  - Also Power8 cores
  - Power8+GPU linked with NVLINK to enable
  - Final numbers gotten from CUDA 8.0, system now has CUDA9, numbers appear similar

GPU Bandwidth Measurements

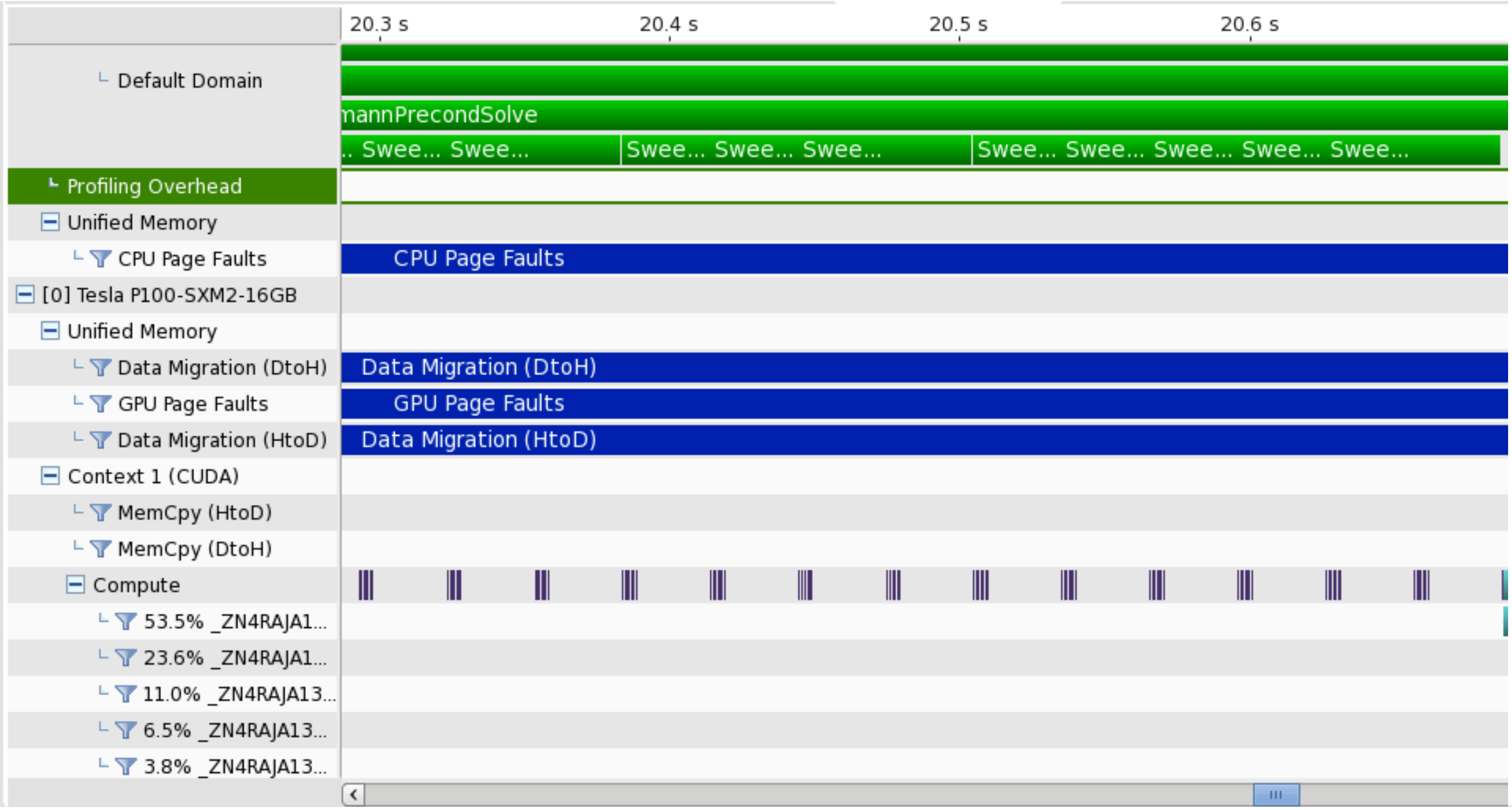


```
======== Profiling result:
======== Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla P100-SXM2-16GB (0)"
    Kernel: _ZN4RAJA13cudaLauncherNINS_26ForallN_IndexTypeConverterIRZN5Armus6Kernel21kGenericElementalLoopIZNS3_6kConstINS2_4Core5Fi
PFvSF_RSD_RKdES5_ISD_E1_SM_NS6_9TypedViewIdPdNS_15LayoutBase_implIN6VarOps16integer_sequenceIJLm0EEEEiEEJiEEEEEvSF_PKcNS6_6SdomIdEiT_
terators16numeric_iteratorIiiPiEEEEEEEEvT_DpT0_
          1                            gld_throughput                    Global Load Throughput  0.00000B/s  0.00000B/s  0.00000B/s
          1                            gst_throughput                   Global Store Throughput  74.909MB/s  74.909MB/s  74.387MB/s
    Kernel: _ZN4RAJA13cudaLauncherNINS_26ForallN_IndexTypeConverterIRZN5Armus6Kernel21kGenericElementalLoopIZNS3_6kConstIN5Ardra8Vari
e_implIN6VarOps16integer_sequenceIJLm0EEEEiEEJiEEEEEvSC_PKcNSA_6SdomIdEiT_OT0_EUnvdl2_PFvSC_SX_SY_iSN_OSV_ES4_ISN_SV_E1_SN_SV_JiEEEJN
        579                            gld_throughput                    Global Load Throughput  0.00000B/s  0.00000B/s  0.00000B/s
        579                            gst_throughput                   Global Store Throughput  357.80MB/s  381.54MB/s  369.07MB/s
```

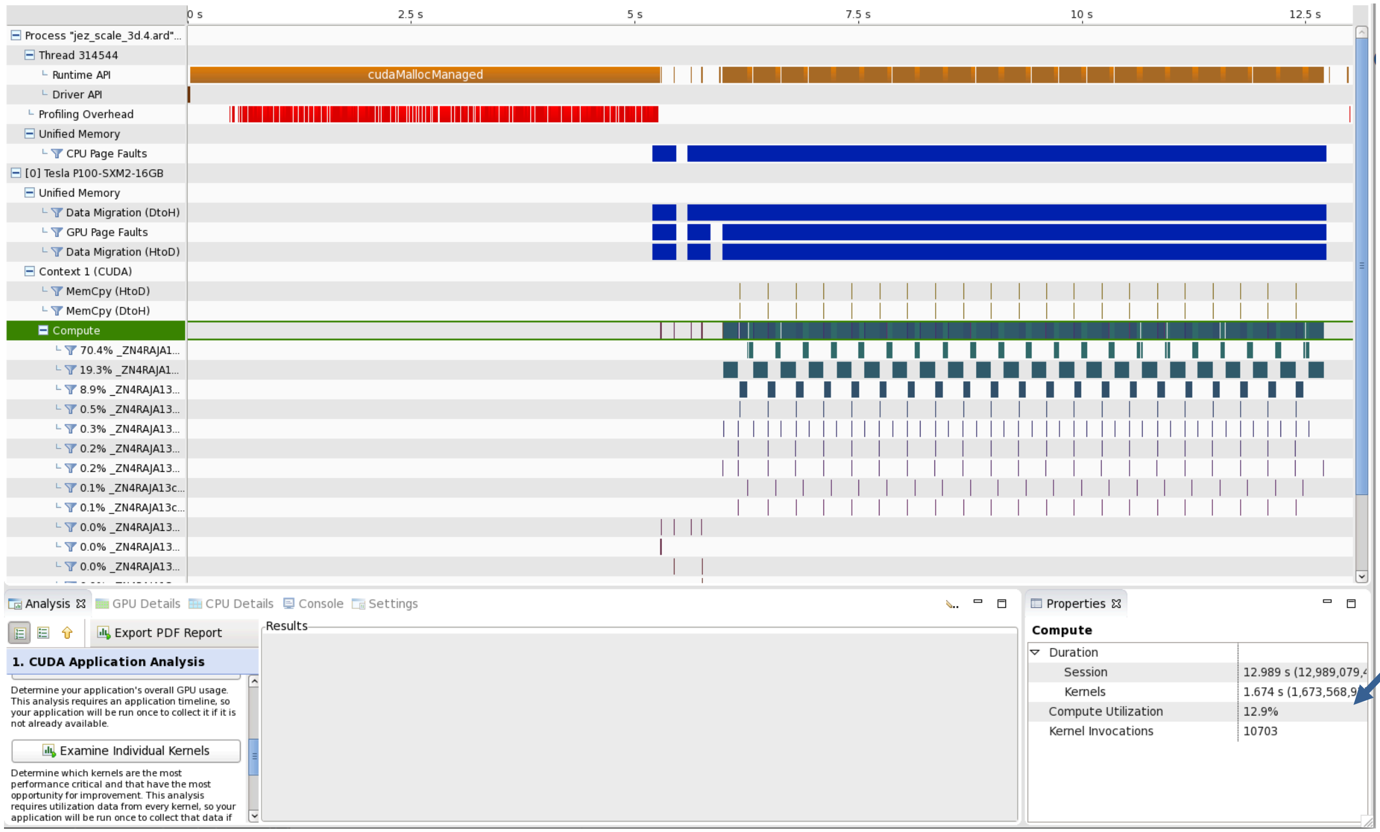
Compute-Dense L0TimesZonal shows good utilization of P100s



Compute-sparse Sweep Shows poor utilization of P100s



Parallel utilization is low across the application, kernels are small



1. CUDA Application Analysis

Determine your application's overall GPU usage. This analysis requires an application timeline, so your application will be run once to collect it if it is not already available.

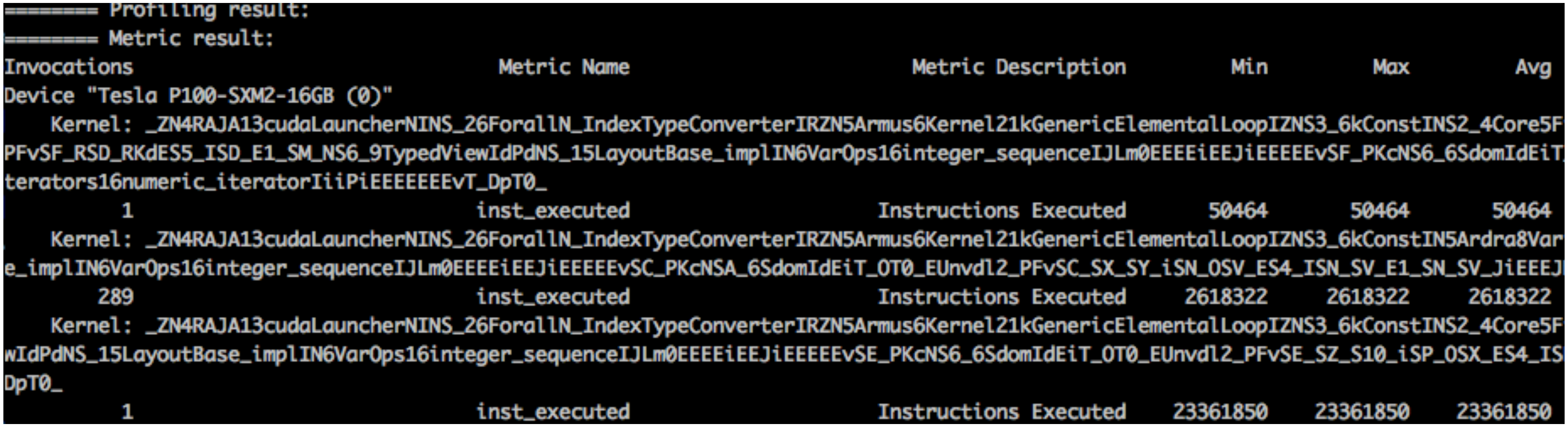
Determine which kernels are the most performance critical and that have the most opportunity for improvement. This analysis requires utilization data from every kernel, so your application will be run once to collect that data if

Compute

Duration	
Session	12.989 s (12,989,079,4
Kernels	1.674 s (1,673,568,9
Compute Utilization	12.9%
Kernel Invocations	10703

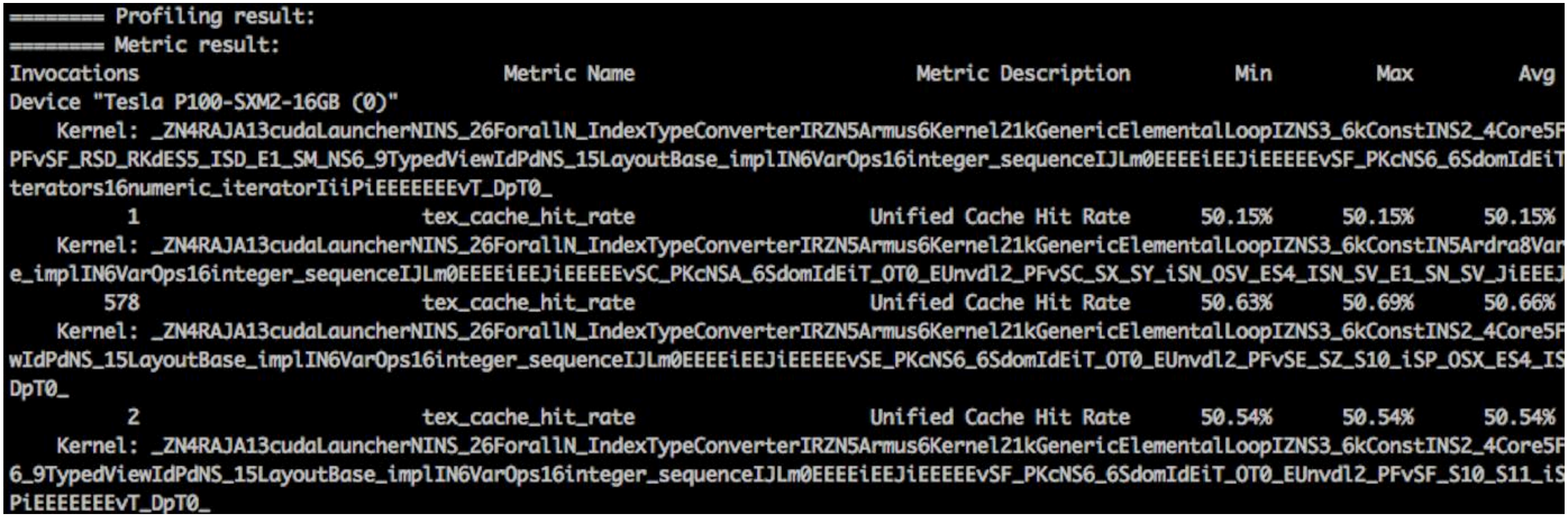
Utilization Numbers

IPC/CPI for GPU



```
========= Profiling result:
========= Metric result:
Invocations                               Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla P100-SXM2-16GB (0)"
    Kernel: _ZN4RAJA13cudaLauncherNINS_26ForallN_IndexTypeConverterIRZN5Armus6Kernel21kGenericElementalLoopIZNS3_6kConstINS2_4Core5F
PFvSF_RSD_RKdES5_ISD_E1_SM_NS6_9TypedViewIdPdNS_15LayoutBase_implIN6VarOps16integer_sequenceIJLm0EEEEiEEJiEEEEEvSF_PKcNS6_6SdomIdEiT
terators16numeric_iteratorIiiPiEEEEEEEEvT_DpT0_
          1                             inst_executed                     Instructions Executed       50464       50464       50464
    Kernel: _ZN4RAJA13cudaLauncherNINS_26ForallN_IndexTypeConverterIRZN5Armus6Kernel21kGenericElementalLoopIZNS3_6kConstIN5Ardra8Var
e_implIN6VarOps16integer_sequenceIJLm0EEEEiEEJiEEEEEvSC_PKcNSA_6SdomIdEiT_OT0_EUnvdl2_PFvSC_SX_SY_iSN_OSV_ES4_ISN_SV_E1_SN_SV_JiEEEJ
        289                             inst_executed                     Instructions Executed     2618322     2618322     2618322
    Kernel: _ZN4RAJA13cudaLauncherNINS_26ForallN_IndexTypeConverterIRZN5Armus6Kernel21kGenericElementalLoopIZNS3_6kConstINS2_4Core5F
wIdPdNS_15LayoutBase_implIN6VarOps16integer_sequenceIJLm0EEEEiEEJiEEEEEvSE_PKcNS6_6SdomIdEiT_OT0_EUnvdl2_PFvSE_SZ_S10_iSP_OSX_ES4_IS
DpT0_
          1                             inst_executed                     Instructions Executed    23361850    23361850    23361850
```

Cache hit rates are relatively good on this problem size



```
========== Profiling result:
========== Metric result:
Invocations                              Metric Name                        Metric Description         Min         Max         Avg
Device "Tesla P100-SXM2-16GB (0)"
    Kernel: _ZN4RAJA13cudaLauncherNINS_26ForallN_IndexTypeConverterIRZN5Armus6Kernel21kGenericElementalLoopIZNS3_6kConstINS2_4Core5F
PFvSF_RSD_RKdES5_ISD_E1_SM_NS6_9TypedViewIdPdNS_15LayoutBase_implIN6VarOps16integer_sequenceIJLm0EEEEiEEJiEEEEEvSF_PKcNS6_6SdomIdEiT
terators16numeric_iteratorIiiPiEEEEEEEvT_DpT0_
          1                tex_cache_hit_rate                   Unified Cache Hit Rate      50.15%      50.15%      50.15%
    Kernel: _ZN4RAJA13cudaLauncherNINS_26ForallN_IndexTypeConverterIRZN5Armus6Kernel21kGenericElementalLoopIZNS3_6kConstIN5Ardra8Var
e_implIN6VarOps16integer_sequenceIJLm0EEEEiEEJiEEEEEvSC_PKcNSA_6SdomIdEiT_OT0_EUnvdl2_PFvSC_SX_SY_iSN_OSV_ES4_ISN_SV_E1_SN_SV_JiEEEJ
        578                tex_cache_hit_rate                   Unified Cache Hit Rate      50.63%      50.69%      50.66%
    Kernel: _ZN4RAJA13cudaLauncherNINS_26ForallN_IndexTypeConverterIRZN5Armus6Kernel21kGenericElementalLoopIZNS3_6kConstINS2_4Core5F
wIdPdNS_15LayoutBase_implIN6VarOps16integer_sequenceIJLm0EEEEiEEJiEEEEEvSE_PKcNS6_6SdomIdEiT_OT0_EUnvdl2_PFvSE_SZ_S10_iSP_OSX_ES4_IS
DpT0_
          2                tex_cache_hit_rate                   Unified Cache Hit Rate      50.54%      50.54%      50.54%
    Kernel: _ZN4RAJA13cudaLauncherNINS_26ForallN_IndexTypeConverterIRZN5Armus6Kernel21kGenericElementalLoopIZNS3_6kConstINS2_4Core5F
6_9TypedViewIdPdNS_15LayoutBase_implIN6VarOps16integer_sequenceIJLm0EEEEiEEJiEEEEEvSF_PKcNS6_6SdomIdEiT_OT0_EUnvdl2_PFvSF_S10_S11_iS
PiEEEEEEEvT_DpT0_
```

Los Alamos

NATIONAL LABORATORY

EST. 1943

Los Alamos
NATIONAL LABORATORY
EST.1943

Delivering science and technology
to protect our nation
and promote world stability

NNSA
National Nuclear Security Administration

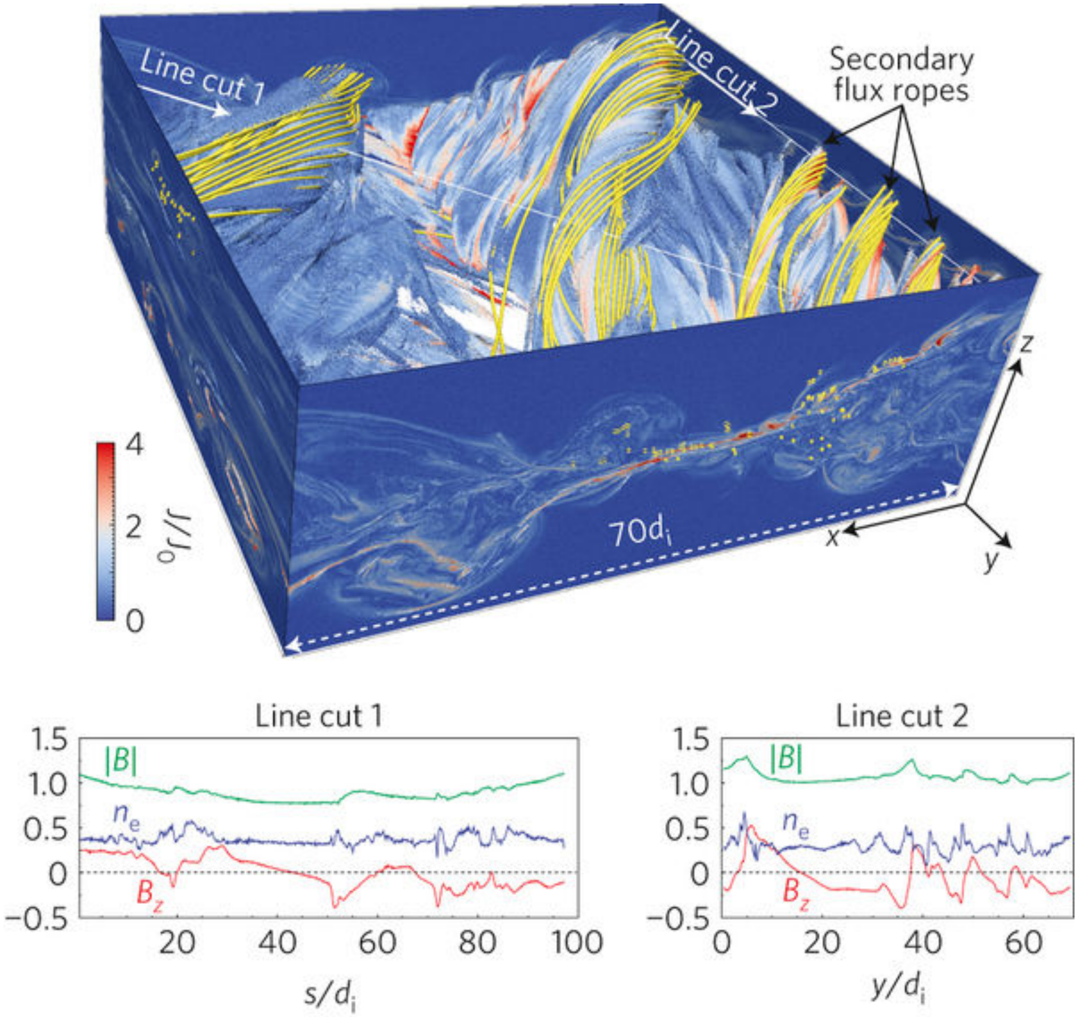
Operated by Los Alamos National Security, LLC for the U.S. Department of Energy's NNSA

Co-Design L2 Milestone Review

Ben Bergen

August 30th, 2017

Operated by Los Alamos National Security, LLC for the U.S. Department of Energy's NNSA

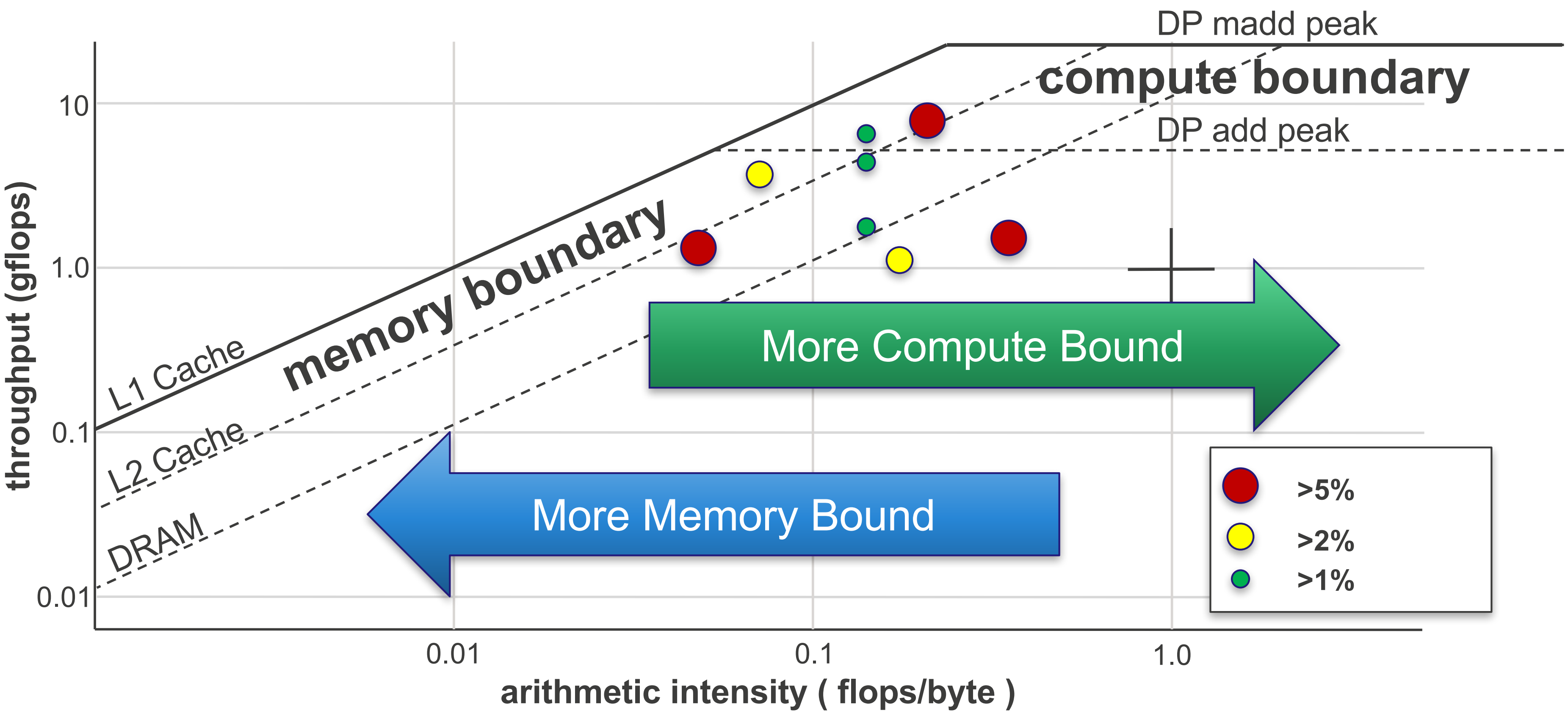


- **Terminology**
- **PARTISN**
- **VPIC**
- **FleCSALE**

Pipeline Hazards

- **Structural Hazard**
  - A hazard that arises from a resource conflict where the hardware cannot support all possible instruction combinations simultaneously
  - Example: the pipeline stalls because a vector addition requires both of its operands to be shuffled, but shuffles can only occur on one execution unit
- **Data Hazard**
  - A hazard that arises when an instruction depends on the result of a previous instruction
  - Example: the pipeline stalls because a load operation from main memory has not yet completed
- **Control Hazard**
  - A hazard that arises from a branch or instruction that changes the program counter

Intel Roofline Model – Explanation



PARTISN
Work done by Randy Baker and Joe Zerr

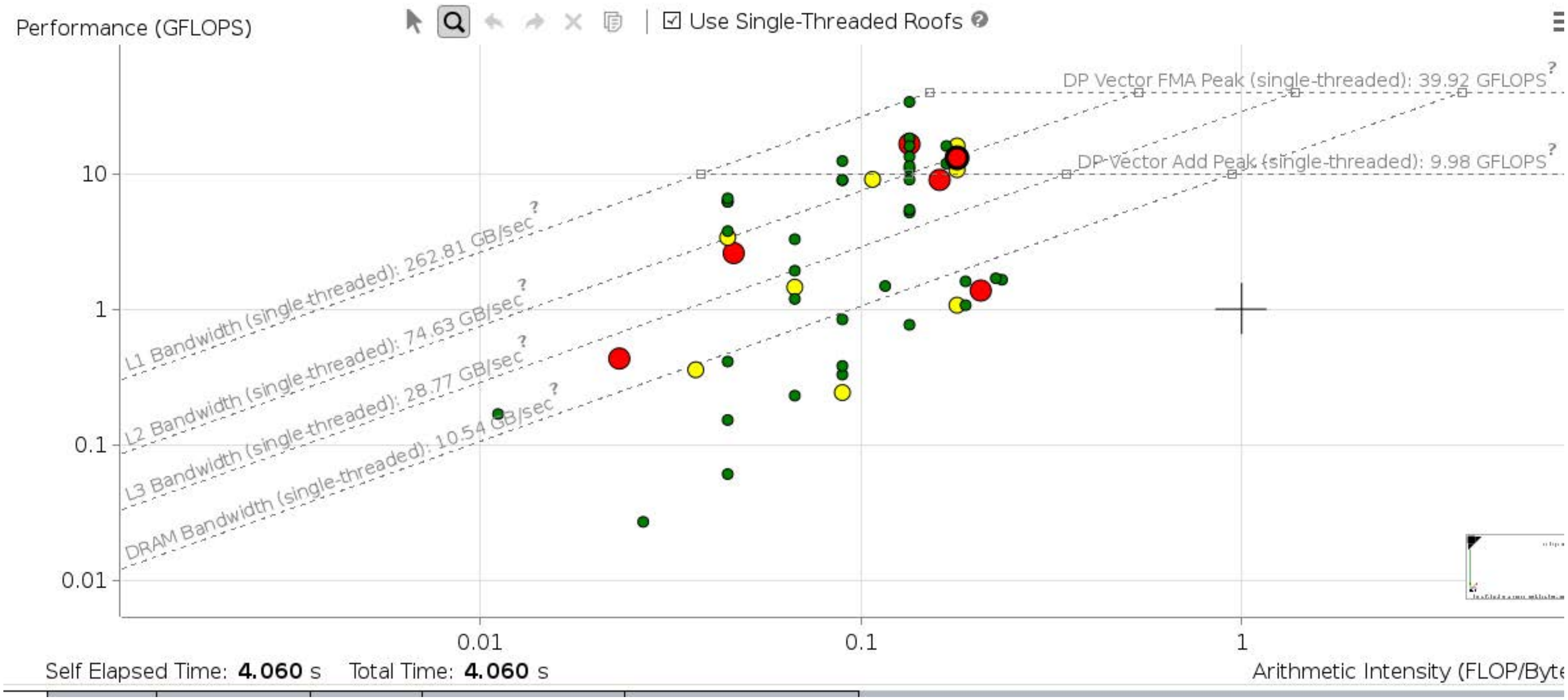
PARTISN
Overview

PARTISN is a time-dependent, multi-dimensional neutron/gamma transport code developed at LANL

- **Basis for SNAP mini-app**
- **110,000 lines of code (Fortran with some C and Python)**
- **Dates to the 1950s/early 1960s**
- **Originally targeted to the CDC-7600**
  - 10 MFlop/s peak
- **Supported by four people (2.5 FTEs)**
  - Methods research
  - Algorithm development
  - Software development & testing
  - User support & applications expertise

PARTISN

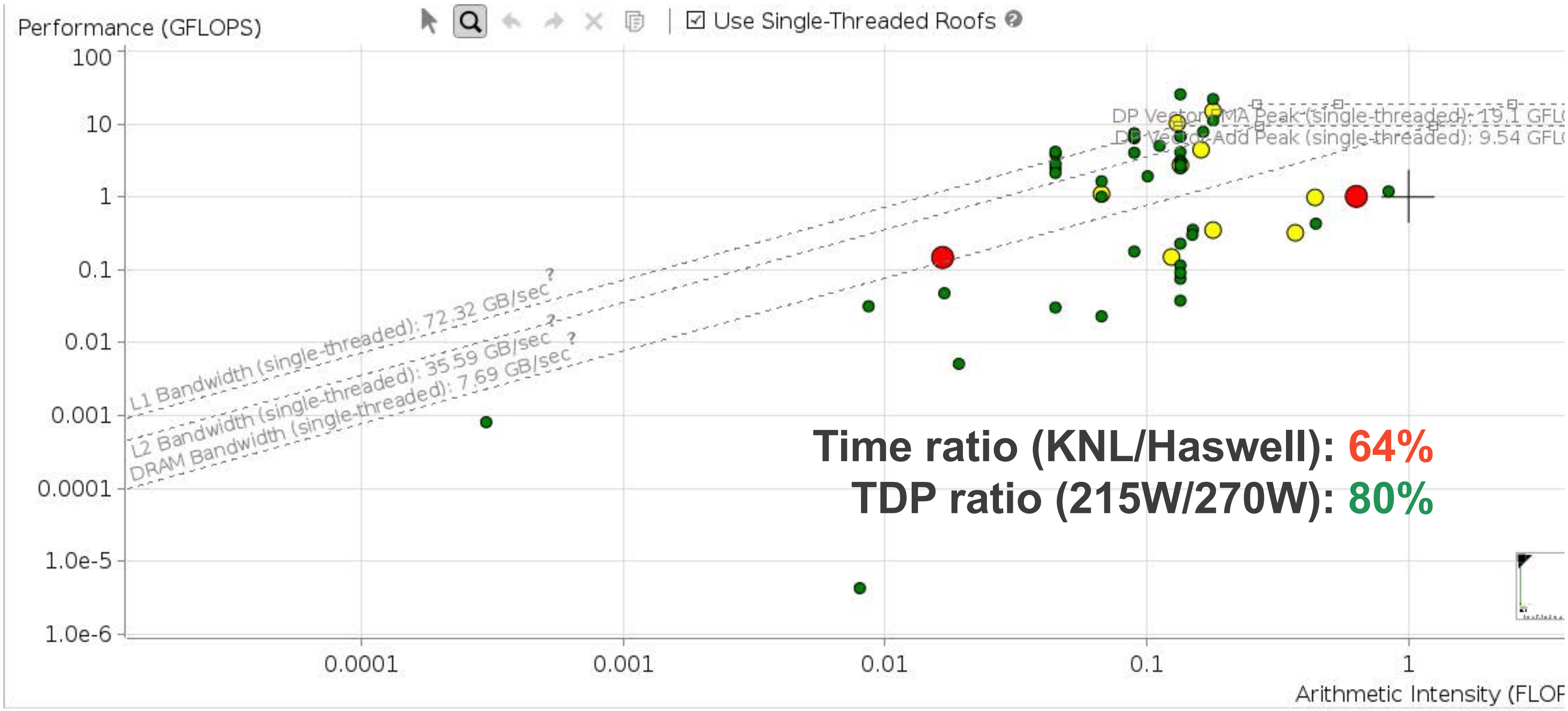
Haswell 32-core, 32 MPI ranks, 32 GB, 48.0 seconds


☑ Use Single-Threaded Roofs

Self Elapsed Time: **4.060** s Total Time: **4.060** s

Performance (GFLOPS)
DP Vector FMA Peak (single-threaded): 39.92 GFLOPS
DP Vector Add Peak (single-threaded): 9.98 GFLOPS
L1 Bandwidth (single-threaded): 262.81 GB/sec
L2 Bandwidth (single-threaded): 74.63 GB/sec
L3 Bandwidth (single-threaded): 28.77 GB/sec
DRAM Bandwidth (single-threaded): 10.54 GB/sec
10
1
0.1
0.01
0.01
0.1
1
Arithmetic Intensity (FLOP/Byte

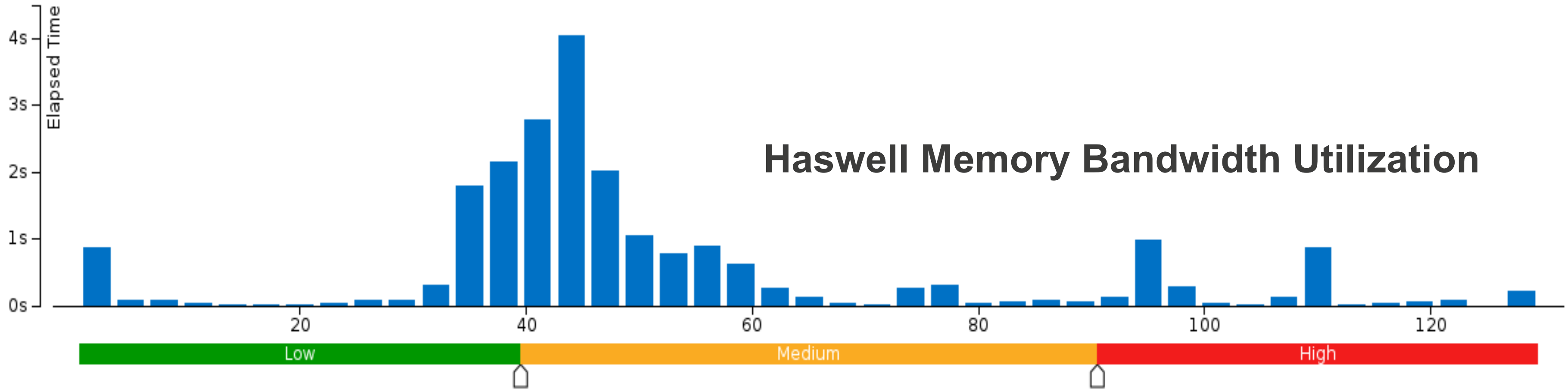
PARTISN
KNL 68-core, 64 MPI ranks, 32 GB, 74.6 seconds



Time ratio (KNL/Haswell): 64%

TDP ratio (215W/270W): 80%

PARTISN
Memory Bandwidth Boundedness

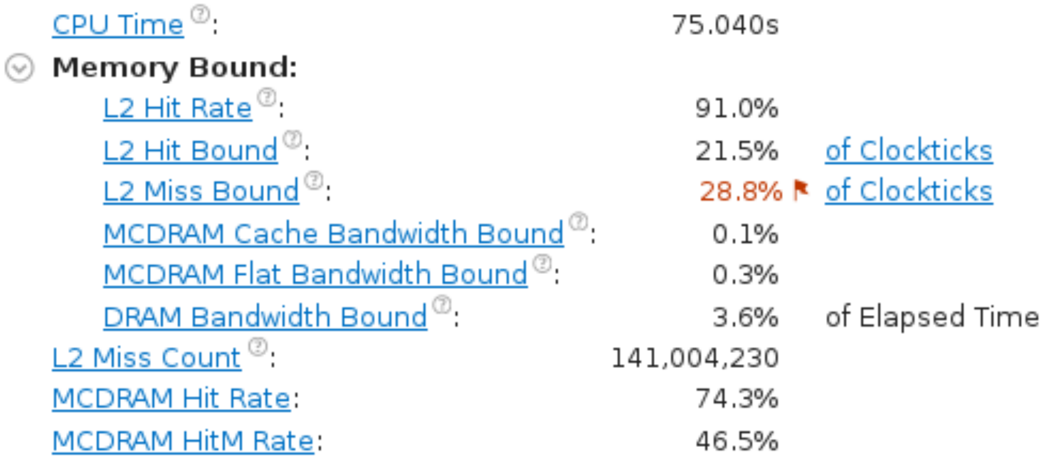


Haswell Memory Bandwidth Utilization

- **PARTISN is only *somewhat* memory bandwidth bound on Haswell**
  - 13.2% DRAM bandwidth bound
  - 0.519 CPI (minimum is 0.25 CPI)
- **Memory bandwidth is less of an issue on KNL**
  - 4.0% DRAM bandwidth bound
  - 1.326 CPI (minimum is 0.5 CPI)

PARTISN
KNL Bottleneck – Prefetch

- **Intel VTune Amplifier**



CPU Time:	75.040s	
Memory Bound:		
L2 Hit Rate:	91.0%	
L2 Hit Bound:	21.5%	of Clockticks
L2 Miss Bound:	28.8% ⚑	of Clockticks
MCDRAM Cache Bandwidth Bound:	0.1%	
MCDRAM Flat Bandwidth Bound:	0.3%	
DRAM Bandwidth Bound:	3.6%	of Elapsed Time
L2 Miss Count:	141,004,230	
MCDRAM Hit Rate:	74.3%	
MCDRAM HitM Rate:	46.5%	

- **Red flag is L2 Miss Bound**
  - In spite of > 90% hit rate
- **Discussion with Intel suggested that compiler-generated prefetch instructions were swamping the prefetcher**

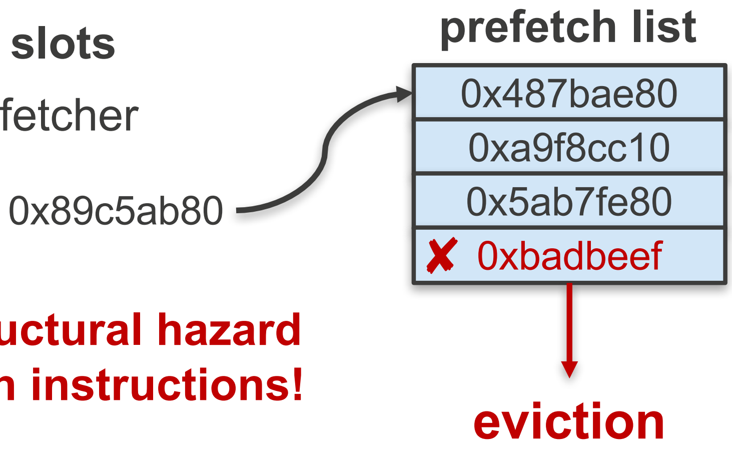
PARTISN
KNL Bottleneck – Prefetch

- **Prefetch instructions can reduce memory latency by requesting data before they are needed, giving more leeway to the dynamic scheduler**

- **KNL supports software and hardware-generated prefetch instructions (priority is given to software-generated prefetch instructions)**

- **The prefetcher has a limited number of available slots**
  - Too many prefetch instructions can *thrash* the prefetcher

- **The likely cause of the KNL bottleneck was a structural hazard caused by too many compiler-generated prefetch instructions!**



prefetch list

0x89c5ab80

0x487bae80

0xa9f8cc10

0x5ab7fe80

✗ 0xbadbeef

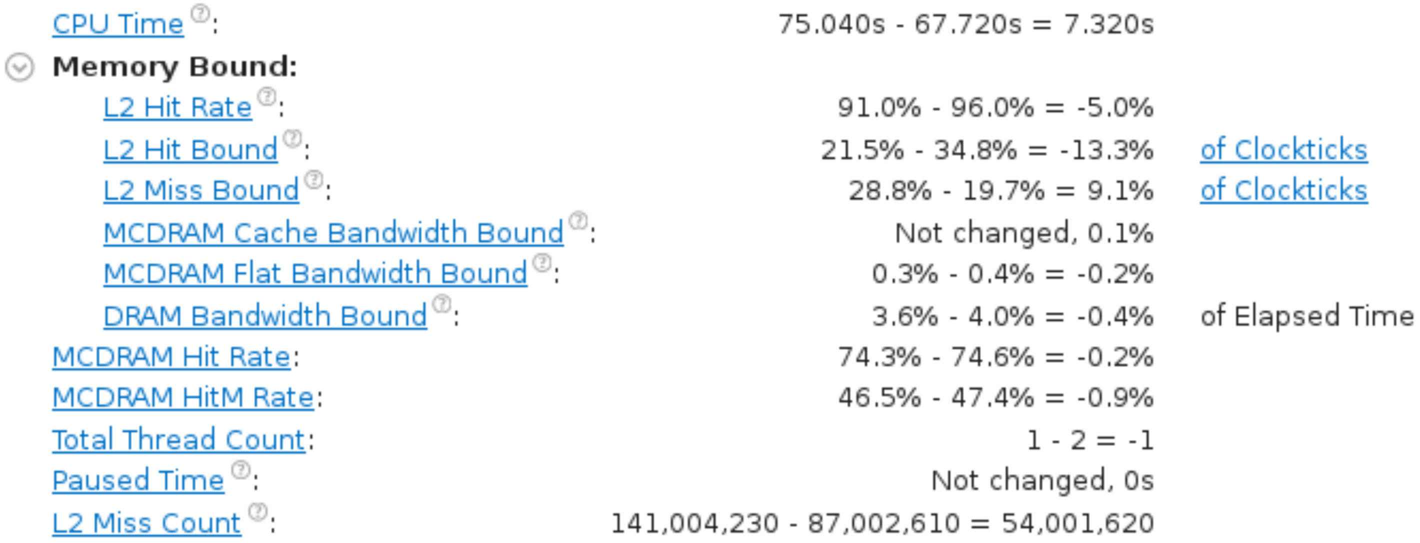
eviction

PARTISN
KNL Optimization – Manual Prefetch

- **Disabled top-level loop prefetch**
  - ***cdir$ noprefetch***
- **Added manual prefetch instructions for key arrays ahead of use**
  - ***call mm_prefetch (q(n,i,j,k), 1)***
  - L2 prefetch is preferable to L1
  - data alignment is not critical
- **Labeled key array as non-temporal, i.e., don't bother caching this…**
  - ***cdir$ vector aligned nontemporal***
  - data alignment is critical
- **Single day of actual coding effort**

PARTISN
Optimizations Significantly Improved KNL Performance

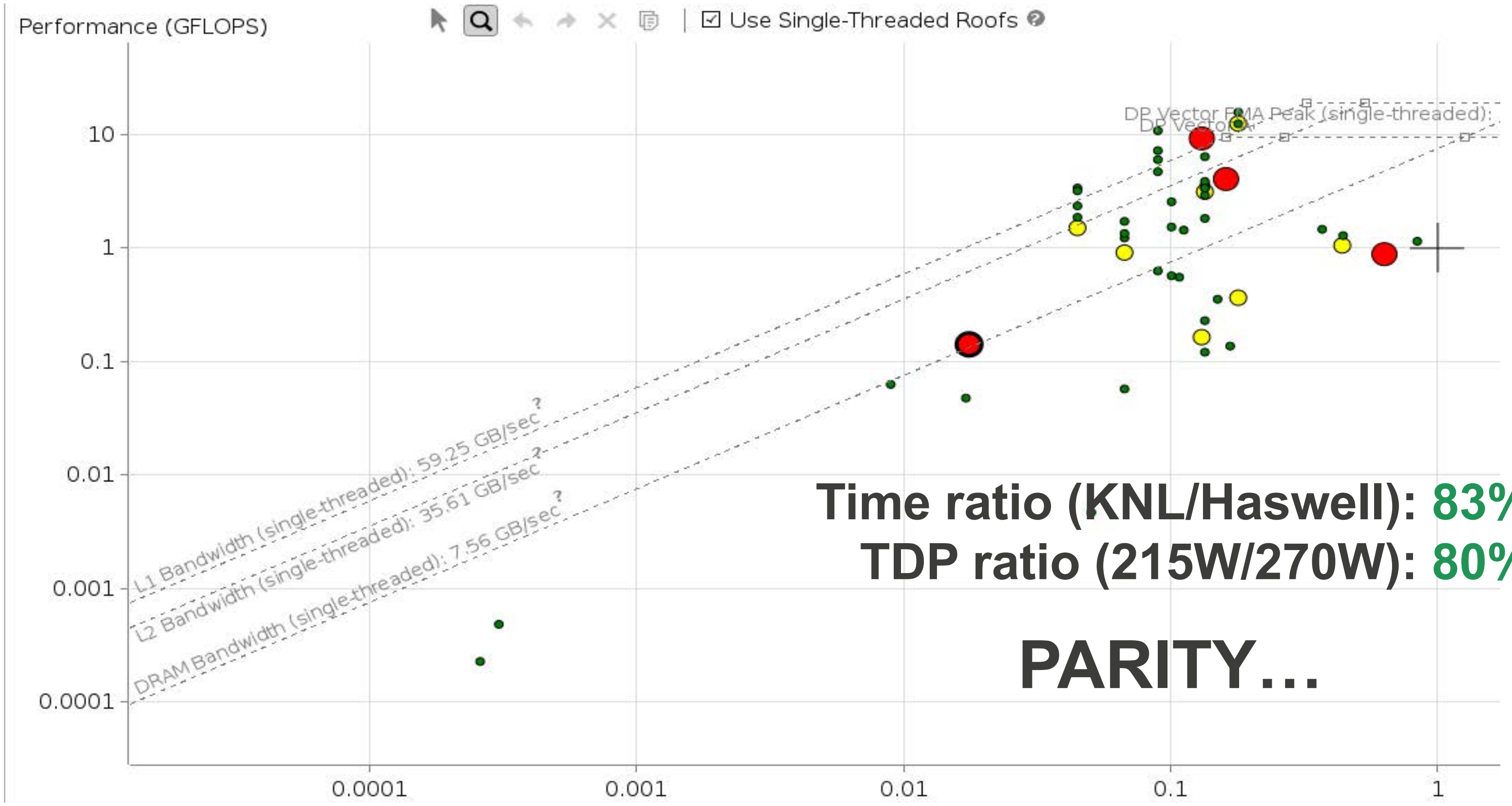
- **Intel VTune Amplifier**



CPU Time:	75.040s - 67.720s = 7.320s	
Memory Bound:		
L2 Hit Rate:	91.0% - 96.0% = -5.0%	
L2 Hit Bound:	21.5% - 34.8% = -13.3%	of Clockticks
L2 Miss Bound:	28.8% - 19.7% = 9.1%	of Clockticks
MCDRAM Cache Bandwidth Bound:	Not changed, 0.1%	
MCDRAM Flat Bandwidth Bound:	0.3% - 0.4% = -0.2%	
DRAM Bandwidth Bound:	3.6% - 4.0% = -0.4%	of Elapsed Time
MCDRAM Hit Rate:	74.3% - 74.6% = -0.2%	
MCDRAM HitM Rate:	46.5% - 47.4% = -0.9%	
Total Thread Count:	1 - 2 = -1	
Paused Time:	Not changed, 0s	
L2 Miss Count:	141,004,230 - 87,002,610 = 54,001,620	

- **L2 Hit Rate**
  - Improved by 5%
- **L2 Miss Bound**
  - Decreased by 9%

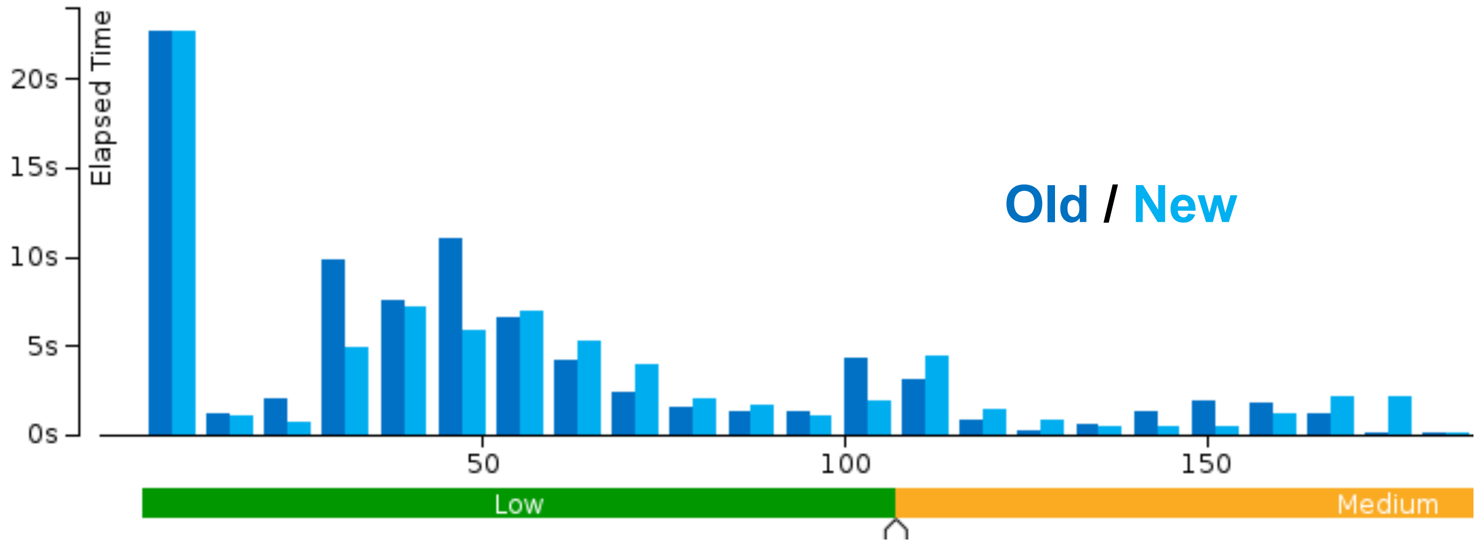
PARTISN
KNL 68-core, 64 MPI ranks, 32 GB, 56.9 seconds



Time ratio (KNL/Haswell): 83%
TDP ratio (215W/270W): 80%

PARITY…

PARTISN
Memory Bandwidth Boundedness



- **PARTISN is *still* only somewhat memory bandwidth bound**
- **The implication is that we still haven’t achieved optimal performance…**
  - We are exploring scheduling issues for nested vector operations

PARTISN
Summary

- **PARTISN is not primarily memory bandwidth bound**
  - KNL performance optimization removed a structural hazard that was affecting memory latency
  - Remaining performance gains are likely to be gotten by improving scheduling of vector operations
- **Coral-EA**
  - Code has been ported to compile on LANL’s pre-Sierra nodes (Power8)
  - Future work will investigate OpenMP function off-load capability

VPIC
Work done by Dave Nystrom

VPIC
Overview

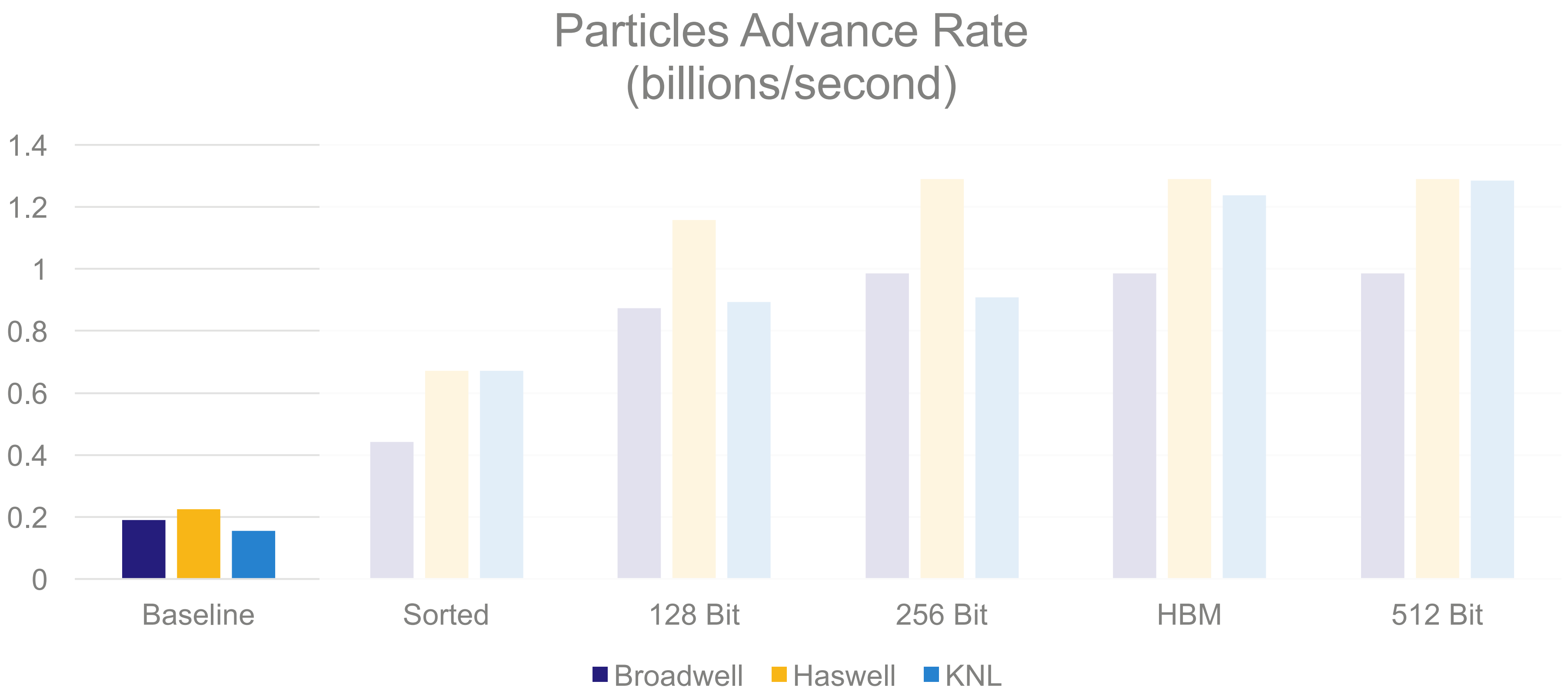
VPIC is a 3D explicit, relativistic, charge-conserving electromagnetic particle-in-cell (PIC) code originally developed by Kevin Bowers at LANL

- Open Source: https://github.com/losalamos/vpic
- Single Precision
- Structured Cartesian Mesh
- Array-of-Structures (AoS) (particle and field data)
- Asynchronous MPI (distributed memory)
- OpenMP or Pthreads (shared memory)
- Explicit short/wide vectorization using hardware intrinsics

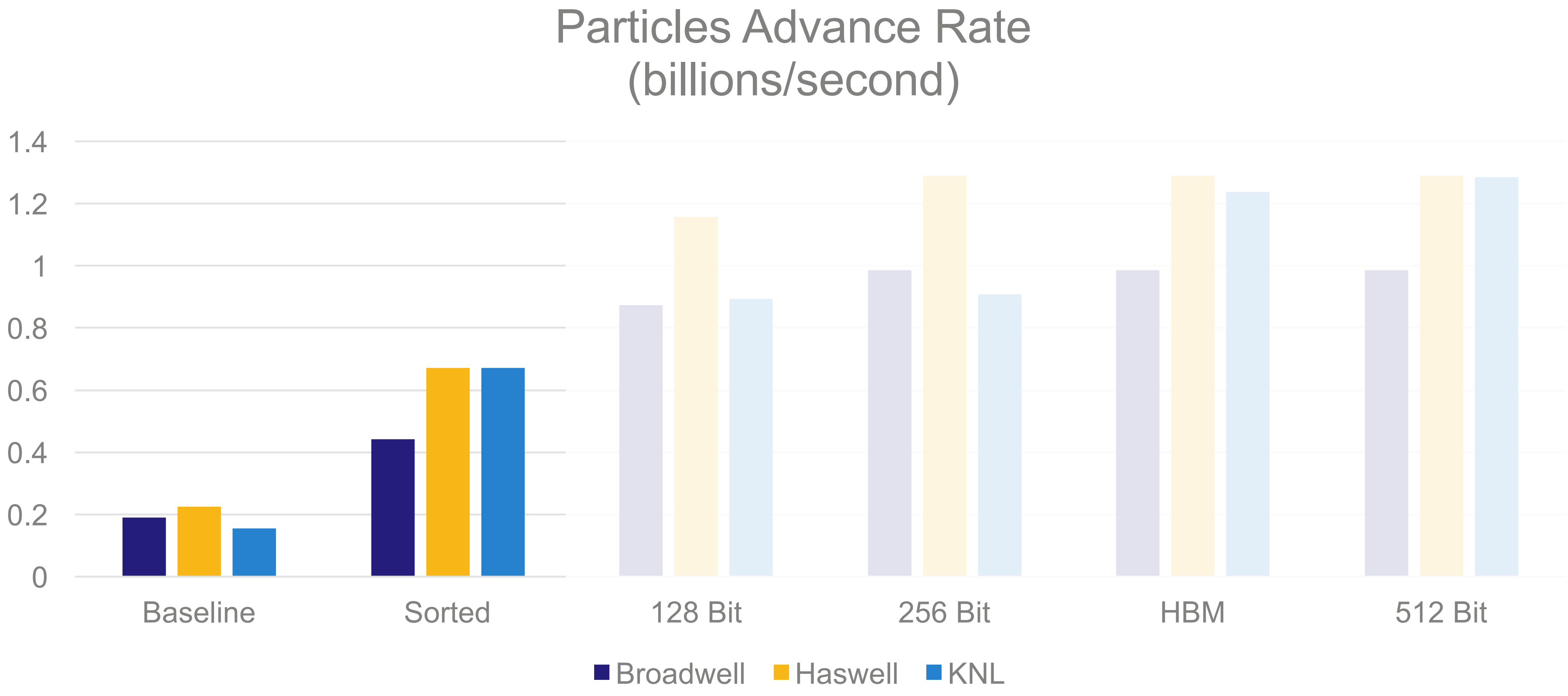
VPIC
Analysis Problems

- **Field Mesh (same for both problems)**
  - 544x96x96 (more cells in the *x*-direction)
  - ~5,000,000 cells
  - Ion and electron species
- **Large Problem (exceeds HBM capacity on KNLs)**
  - 250 particles per cell
  - ~80 GB Memory
- **Small Problem (fits HBM capacity on KNLs)**
  - 25 particles per cell
  - ~8 GB Memory

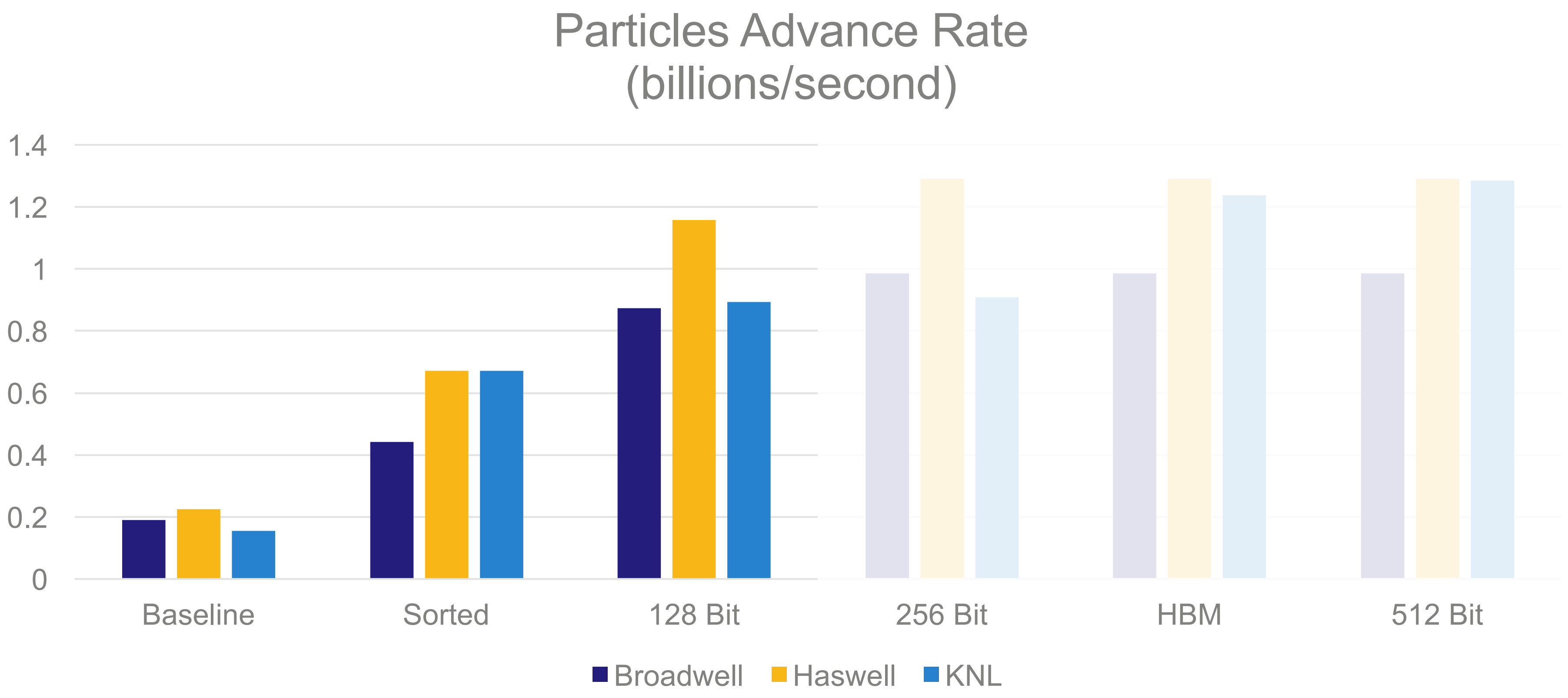
VPIC
Performance – Large Problem



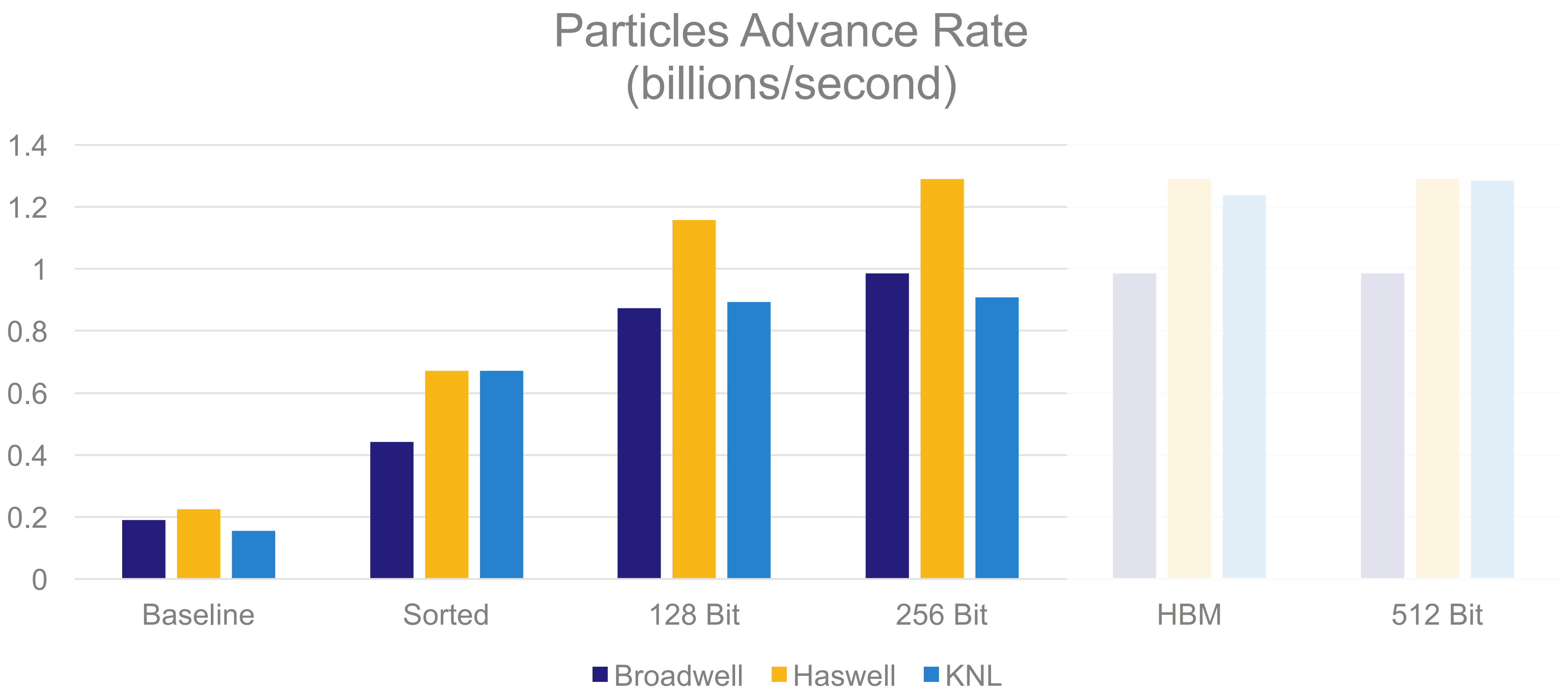
VPIC
Performance – Large Problem



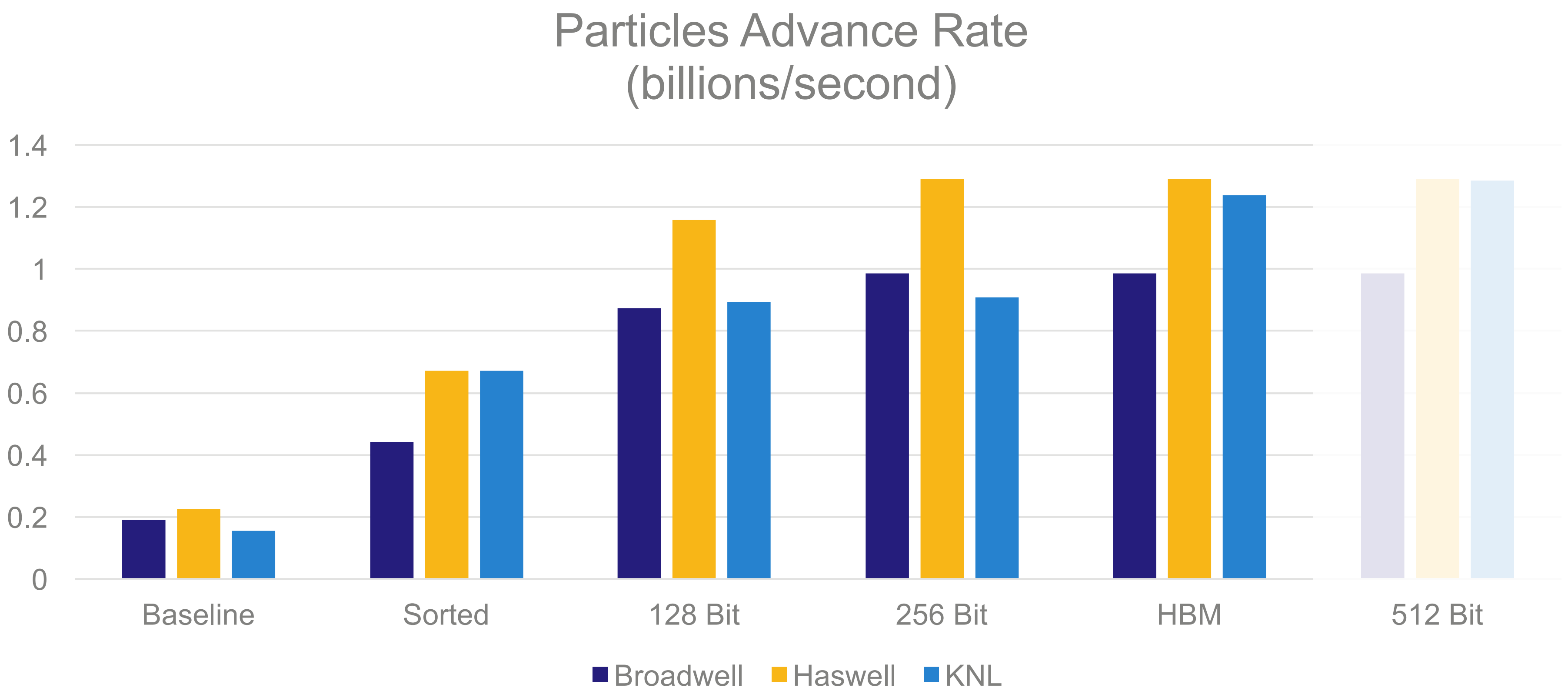
VPIC
Performance – Large Problem



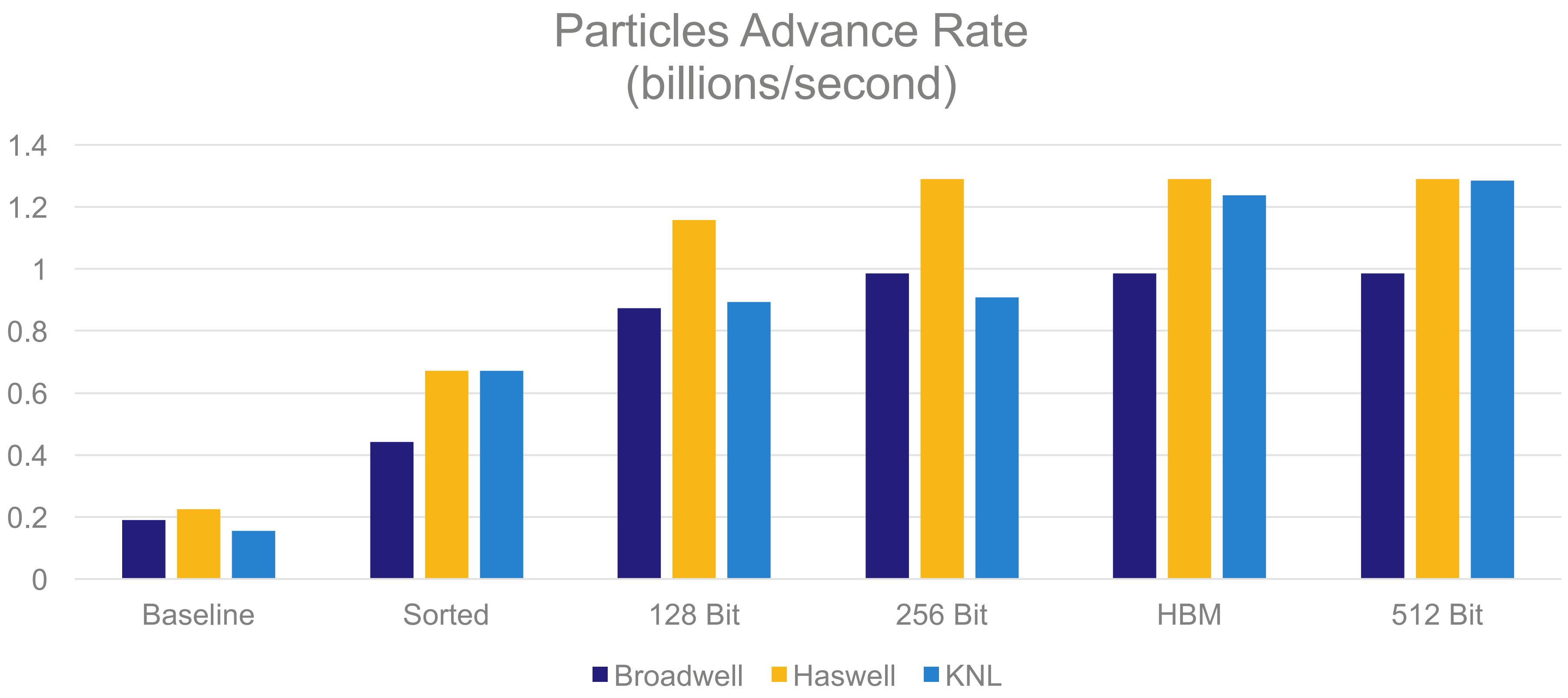
VPIC
Performance – Large Problem



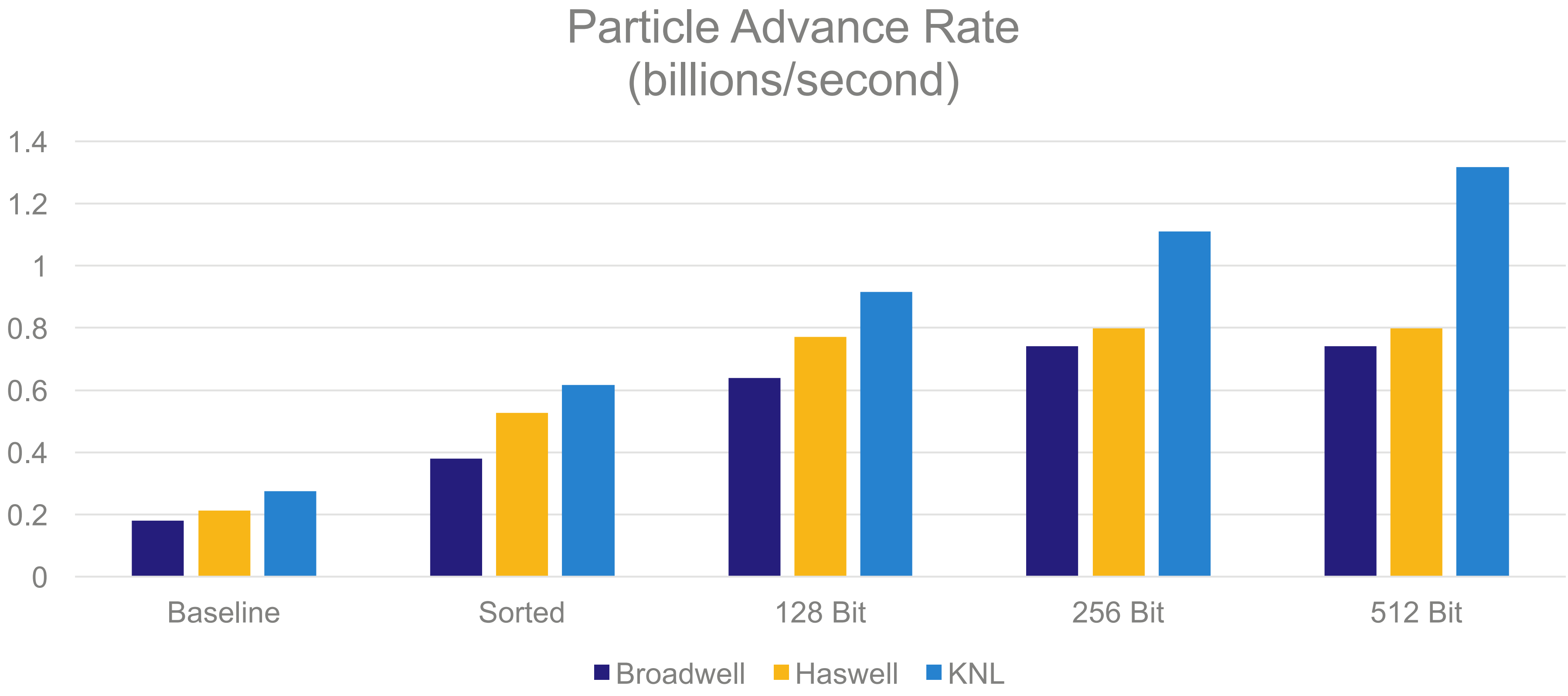
VPIC
Performance – Large Problem



VPIC
Performance – Large Problem

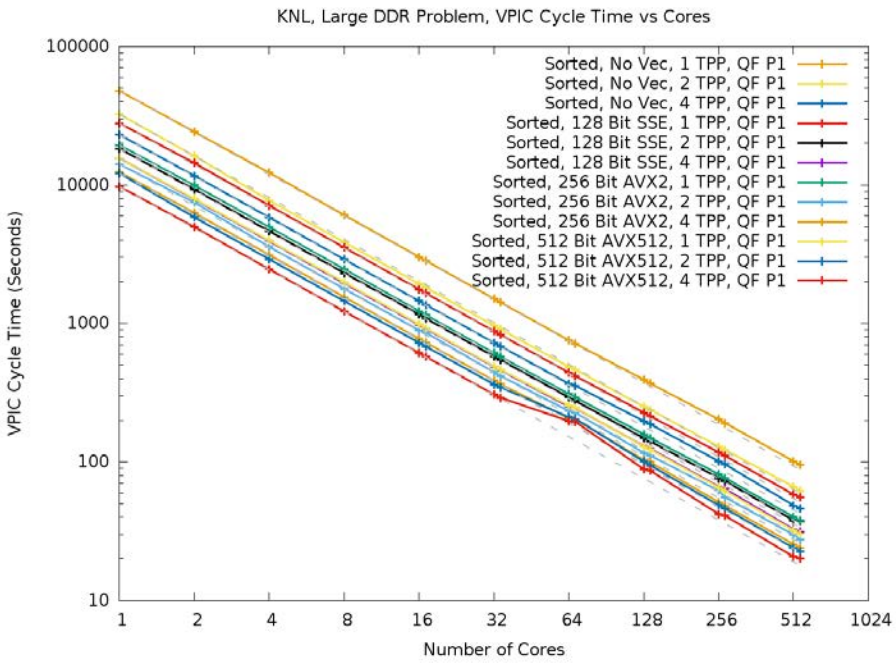


VPIC
Performance – Small Problem

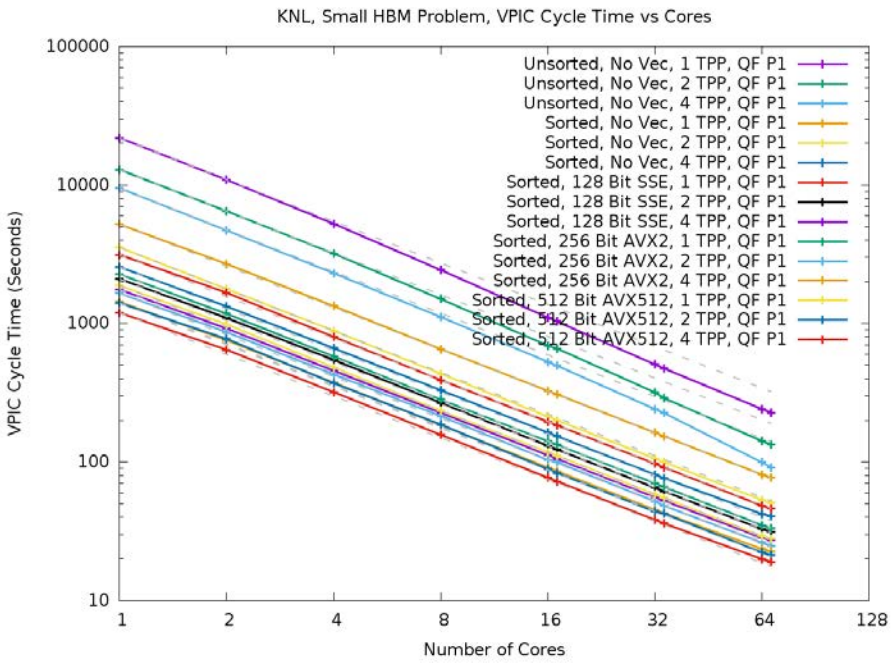


VPIC
Strong Scalability

Large Problem

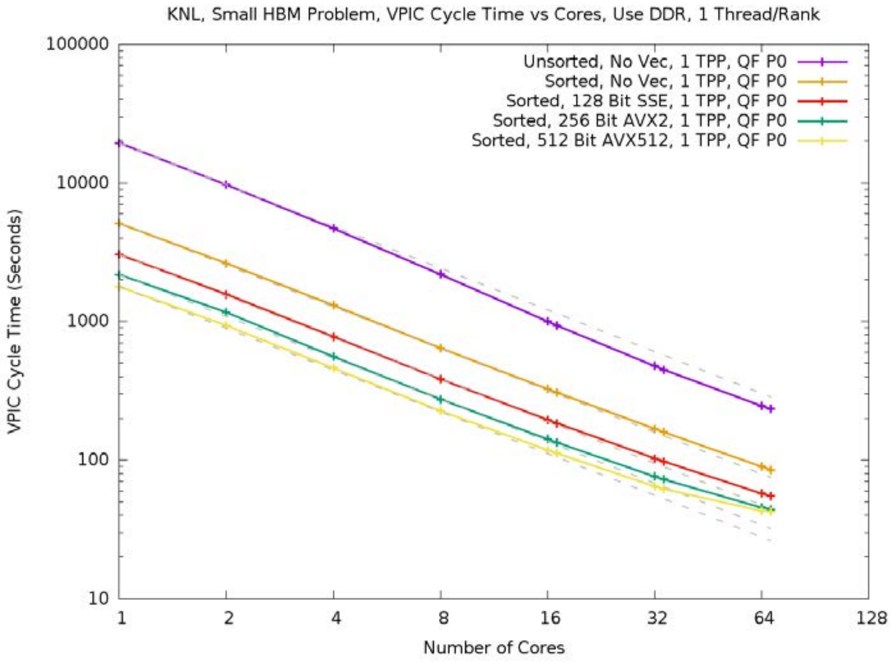


Small Problem

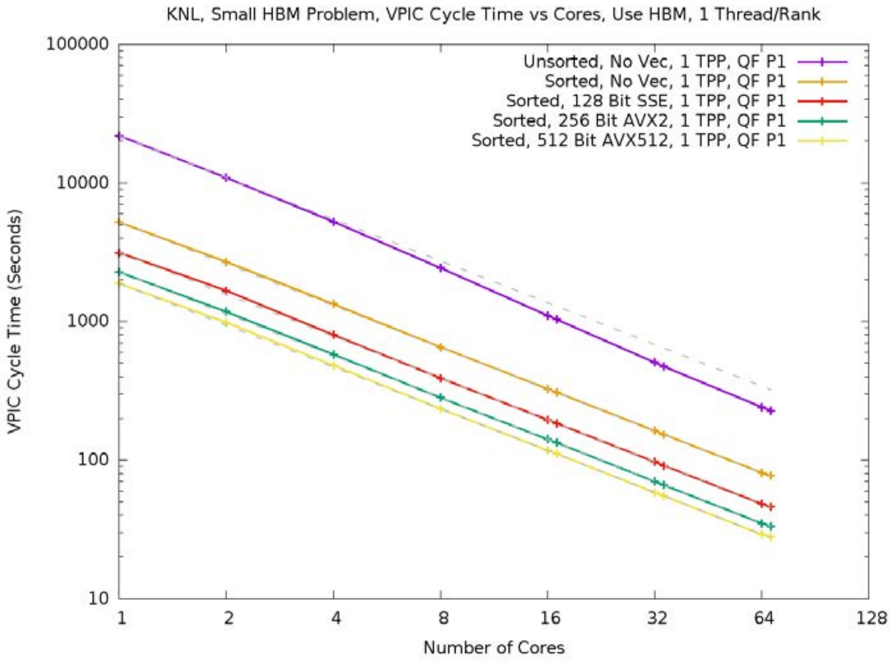


VPIC
Memory Bandwidth Limitations – Small Problem

DDR Memory

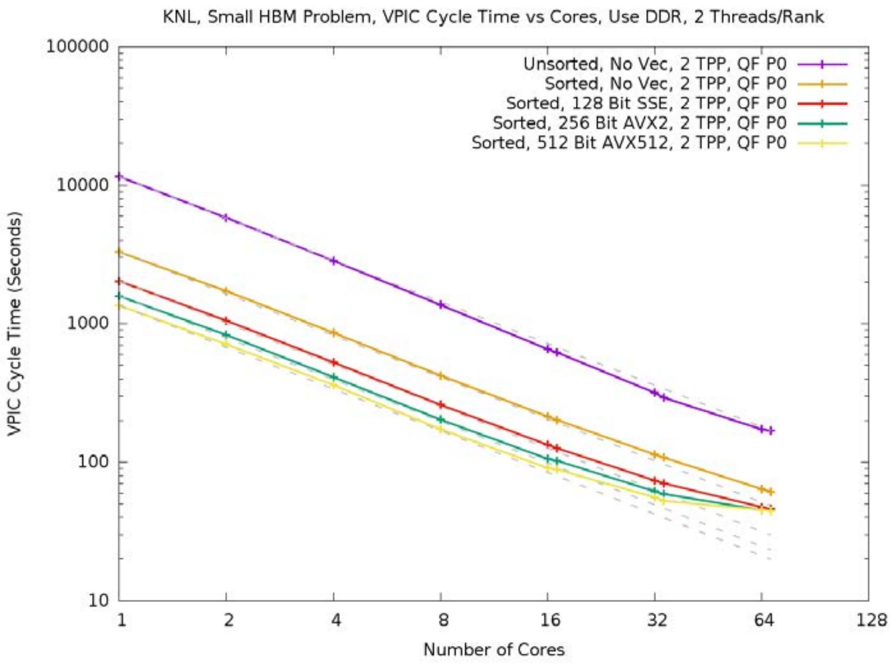


HBM Memory

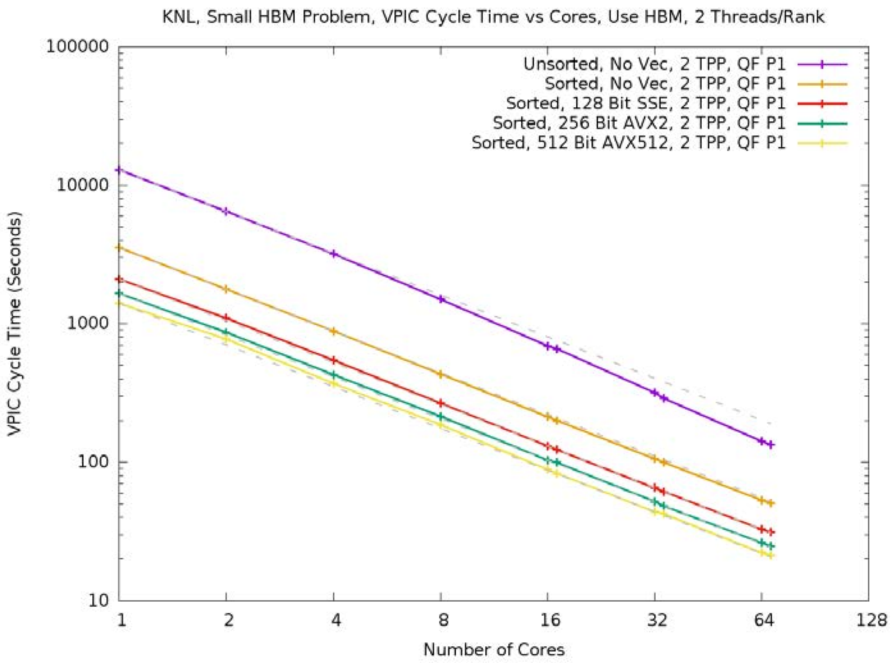


VPIC
Memory Bandwidth Limitations – Small Problem

DDR Memory

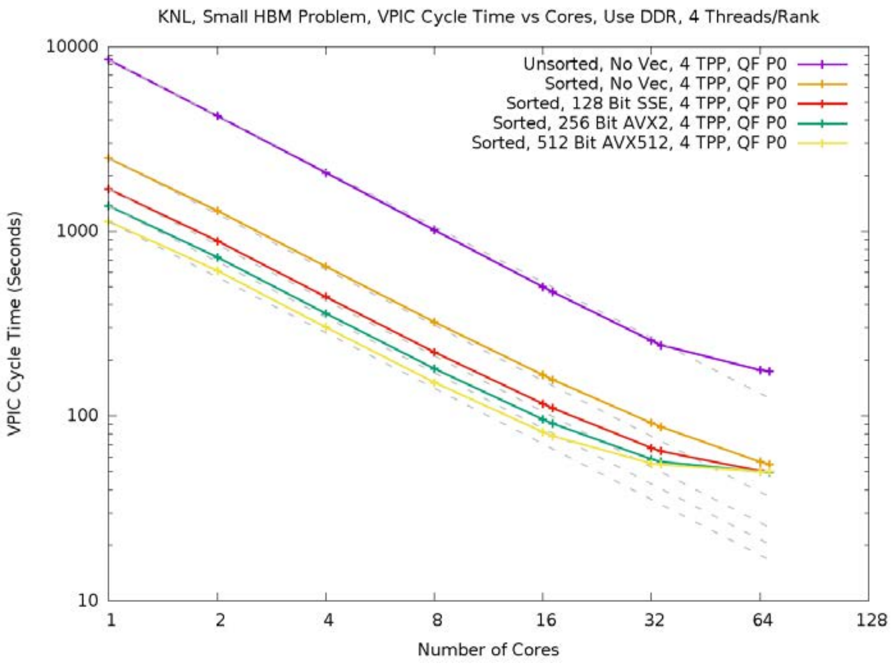


HBM Memory

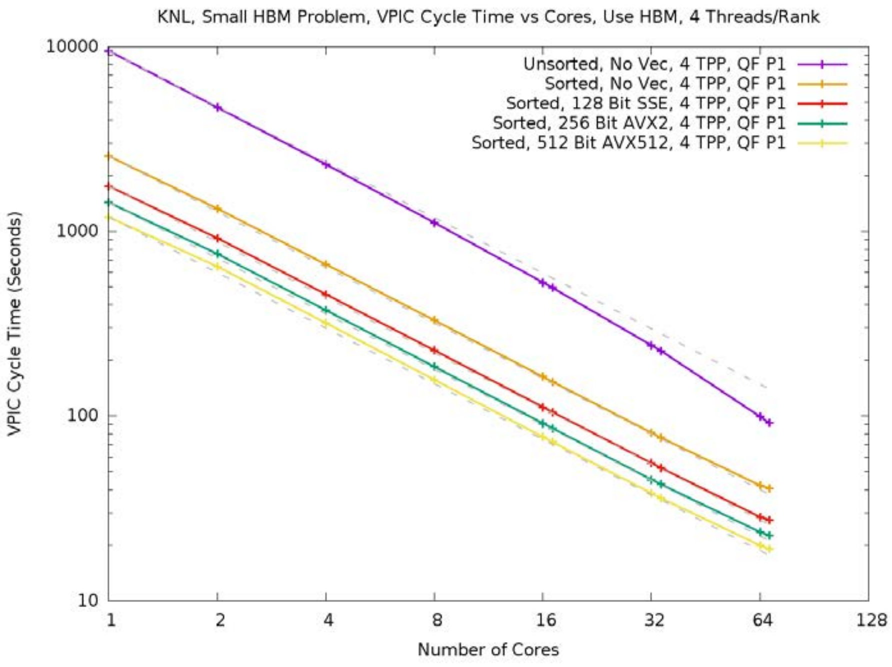


VPIC
Memory Bandwidth Limitations – Small Problem

DDR Memory

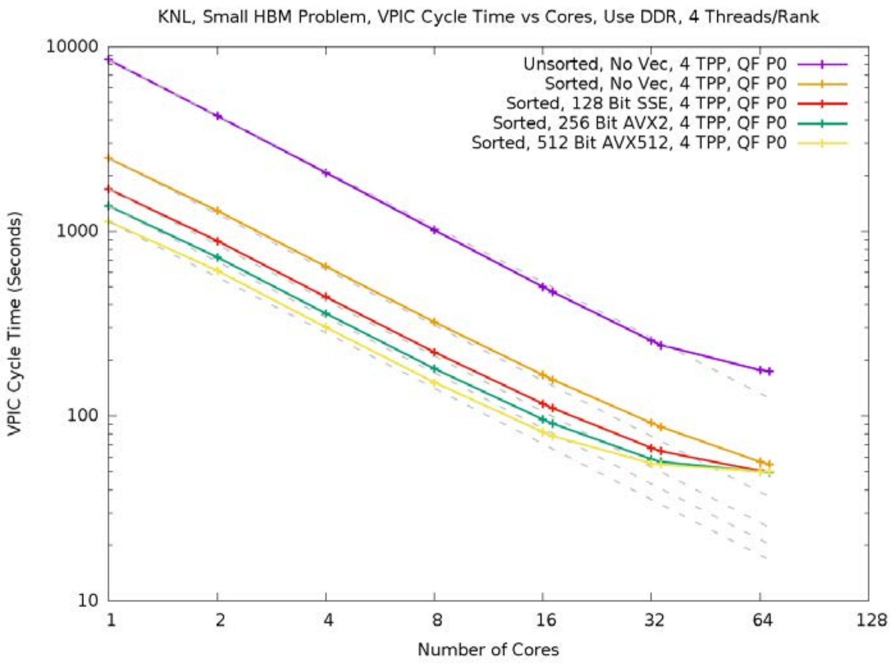


HBM Memory

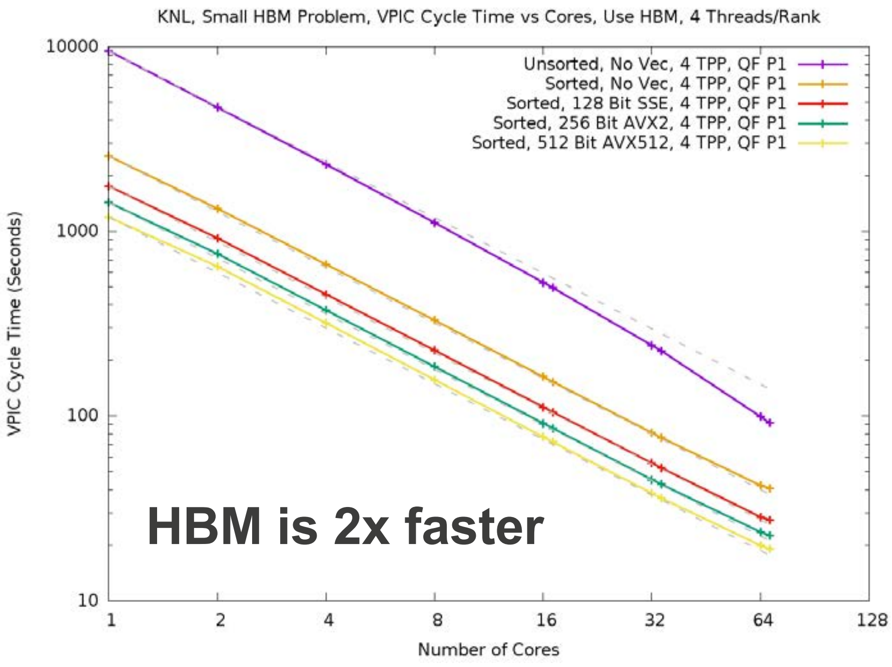


VPIC
Memory Bandwidth Limitations – Small Problem

DDR Memory

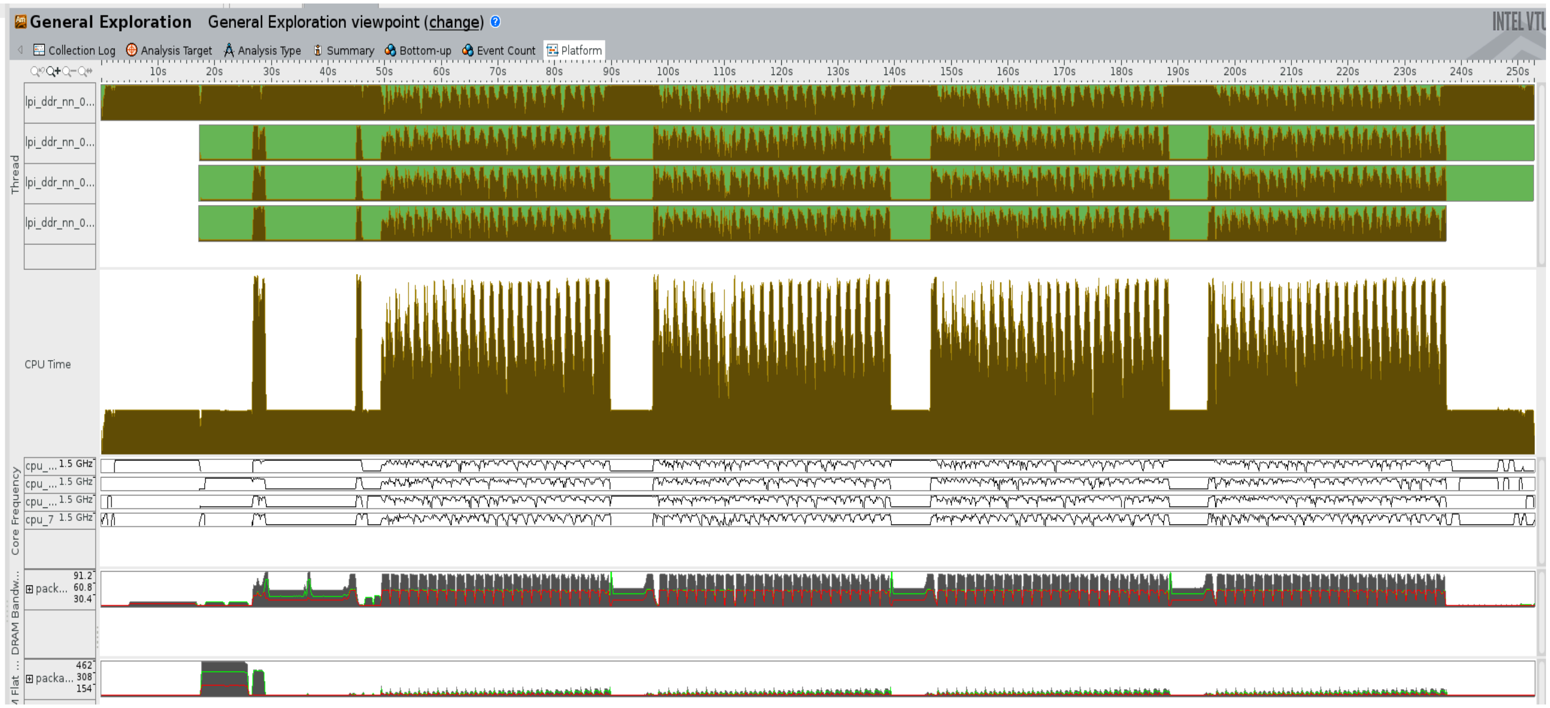


HBM Memory



HBM is 2x faster

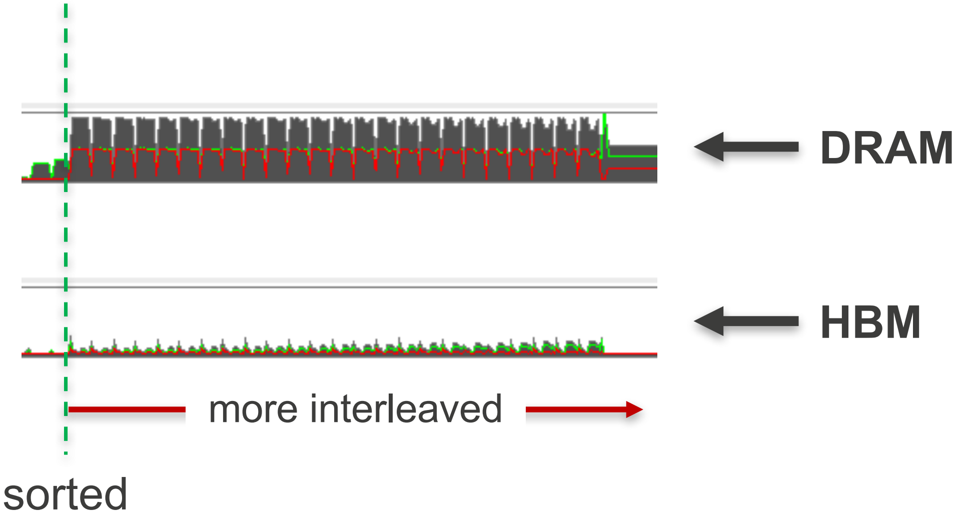
VPIC
Memory Bandwidth Utilization – Large Problem



DRAM

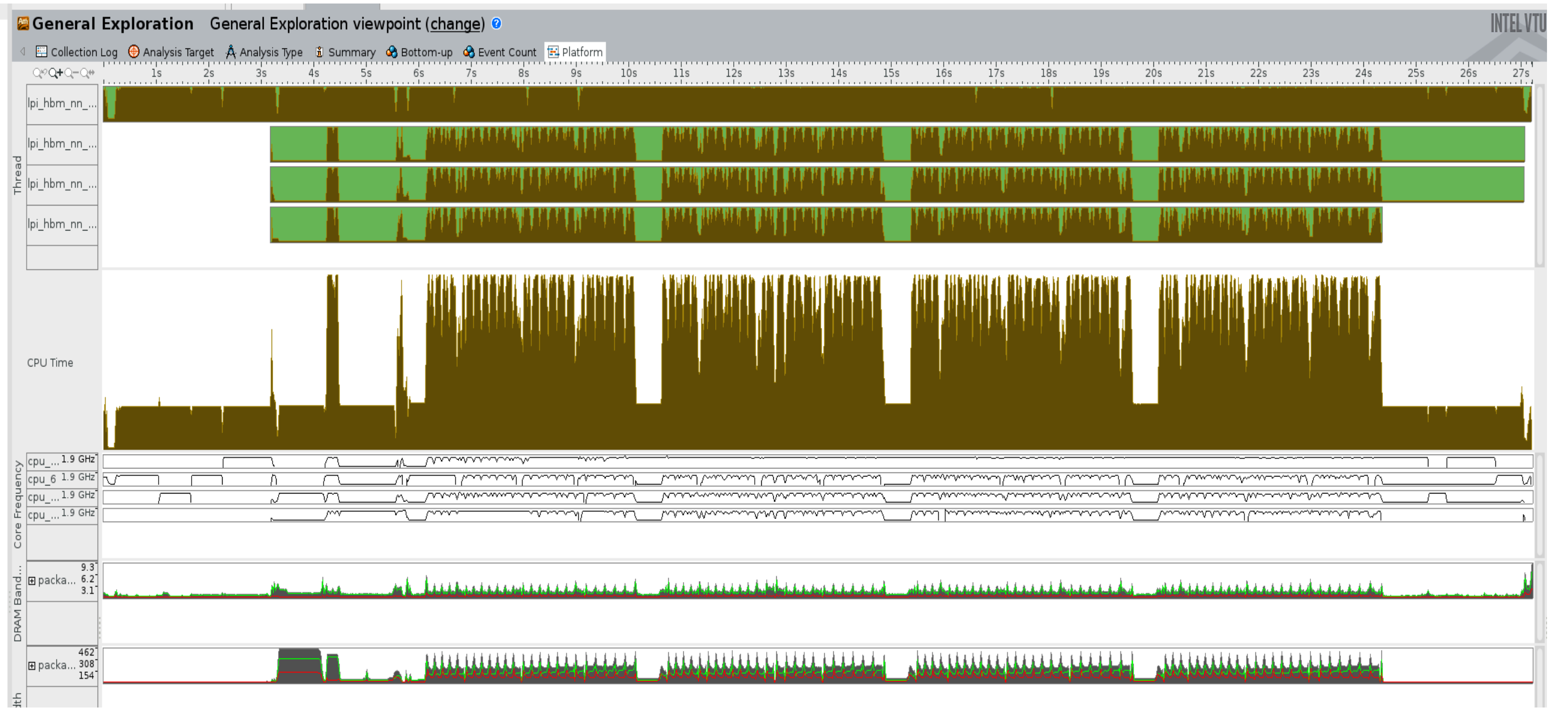
HBM

VPIC
Memory Bandwidth Utilization – Large Problem



- **Sorted particles essentially get full memory bandwidth utilization**
- **As interleaving occurs, HBM utilization increases due to field data accesses**

VPIC
Memory Bandwidth Utilization – Small Problem



DRAM

HBM

VPIC
Port Utilization

Ports In Use	Cycles Fraction
0	64.7%
1	21.6%
2	15.4%
3+	19.4%

- **For 21.6% of the cycles, the CPU executed only 1 µop on any port**
- **Discussion with Intel suggested that this may be a structural hazard caused by shuffle operations required to re-order data for vector arithmetic**

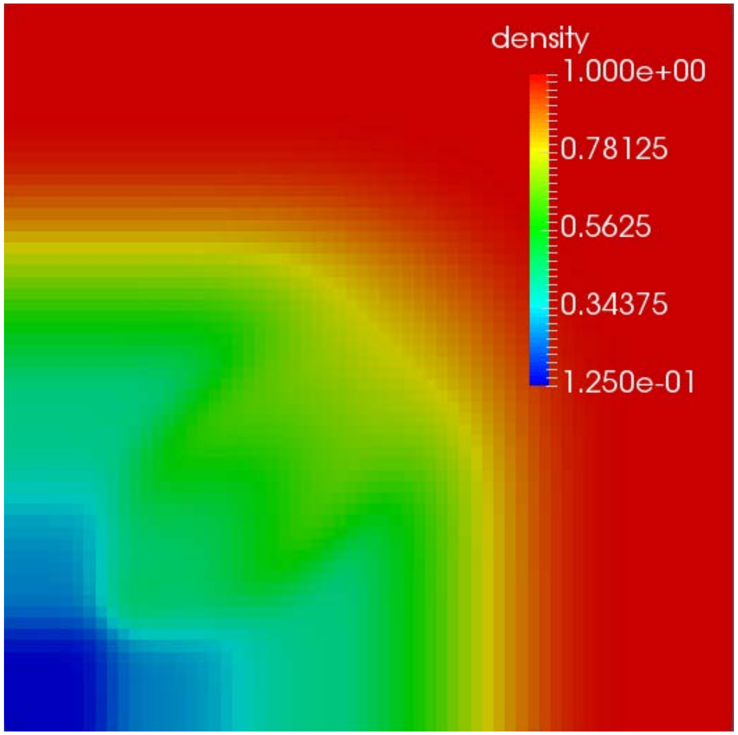
VPIC
Summary

- **Large problem is currently memory-bandwidth limited**
- **Small problem is limited by instruction latency**
- **We have several strategies to improve HBM bandwidth utilization**
  - Re-implementation of transpose operation to use different intrinsics
  - Try gather/scatter support to re-order data by hand
  - Change data layout: Array-of-Structure-of-Vector (AoSoV)
- **Better HBM utilization may provide an impetus to try triple buffering approach**

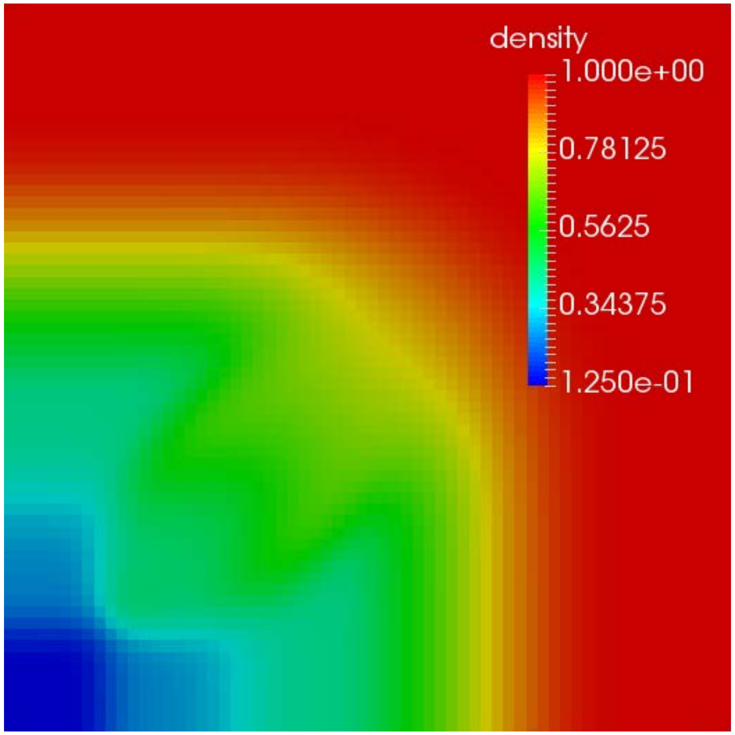
FleCSALE
Work done by Marc Charest

FleCSALE
Runtime Portability

FleCSI allows switch between MPI and Legion runtime models with no change to application code



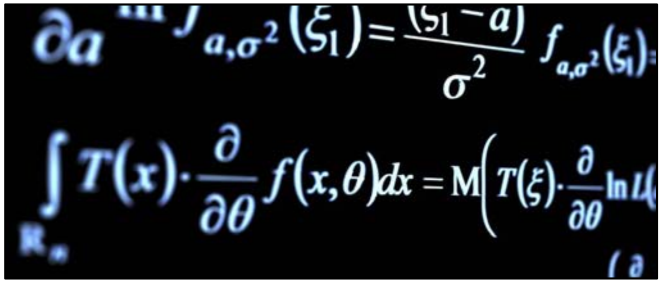
MPI

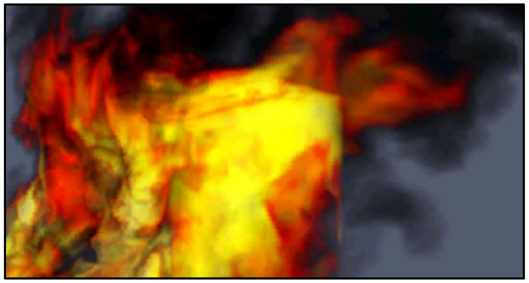


Legion

Exceptional service in the national interest

Sandia National Laboratories


$\int_{\mathbb{R}_n} T(x)\cdot\frac{\partial}{\partial\theta}f(x,\theta)dx = \mathrm{M}\left(T(\xi)\cdot\frac{\partial}{\partial\theta}\ln L\right.$



Metrics for Sandia Applications

S.D. Hammond, C.T. Vaughan, P.T. Lin, D.C. Dinge, C. Hughes, J. Cook, D.M. Pase and R.J. Hoekstra

Application Performance Team (APT), Center for Scientific Computing

Sandia National Laboratories/NM

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Outline

- FY17 has been a very busy year for the APT team
  - Analysis of ATDM EMPIRE/Drekar code
  - Analysis of ATDM SPARC code (using native and Trilinos solvers)
  - Support for IC/NALU and ATDM SPARC runs on Trinity Phase-II (KNL) partition
  - Vendor collaborations (see a few examples later in this talk but many more)
  - ASC Advanced Testbed bring up for ARM/Cavium and Intel Sky Lake
  - General ASC/IC (SIERRA & RAMSES) and research code support
  - Profiling and Debug Tools Testing (and feedback to vendors/ISVs)
  - Continued development of LMDS/OVIS Continuous Performance Monitoring

- Slide deck contains some results from many of these activities
- Focus here on a select few “stories” less than specific metrics (see additional slides for more details), huge effort on behalf of the team to cover many areas (thank you!)

Analysis “Stories”

- In today’s talk we are going to talk about two codesign studies from this year (based on metrics we gathered):
  - **MPI NIC Resource Requirements** – how big do some significant structures need to be?
  - **Compiler Vectorization Levels** – how do multiple generations of compilers compare and what level of vectorization are we getting?

- We have collected lots more metrics/data on codes and have them in the annotated slides for the review committee
  - See the included slides and let us know if you have any questions
  - Work continues to drill down further on these codes and we will be adding continuous performance monitoring and KokkosP-based overnight testing to our suites in FY18

Sandia National Laboratories

APPLICATION MPI/NIC METRICS

Codesign of Next-Generation Network Interconnects

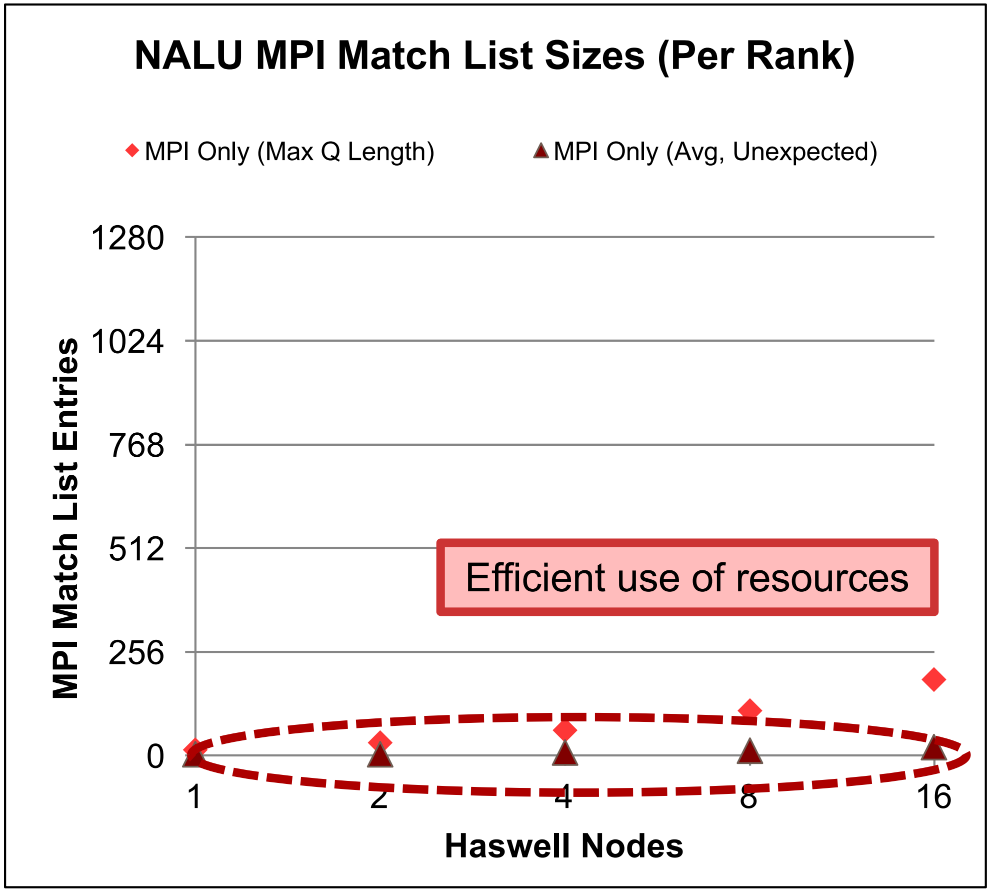
MPI Receive Lists (with Cray)

- Working with Cray interconnect team to evaluate *application* requirements for NIC and MPI acceleration
  - MPI Receive Queues (how many pending receives per rank)
  - Average lengths of unexpected queues

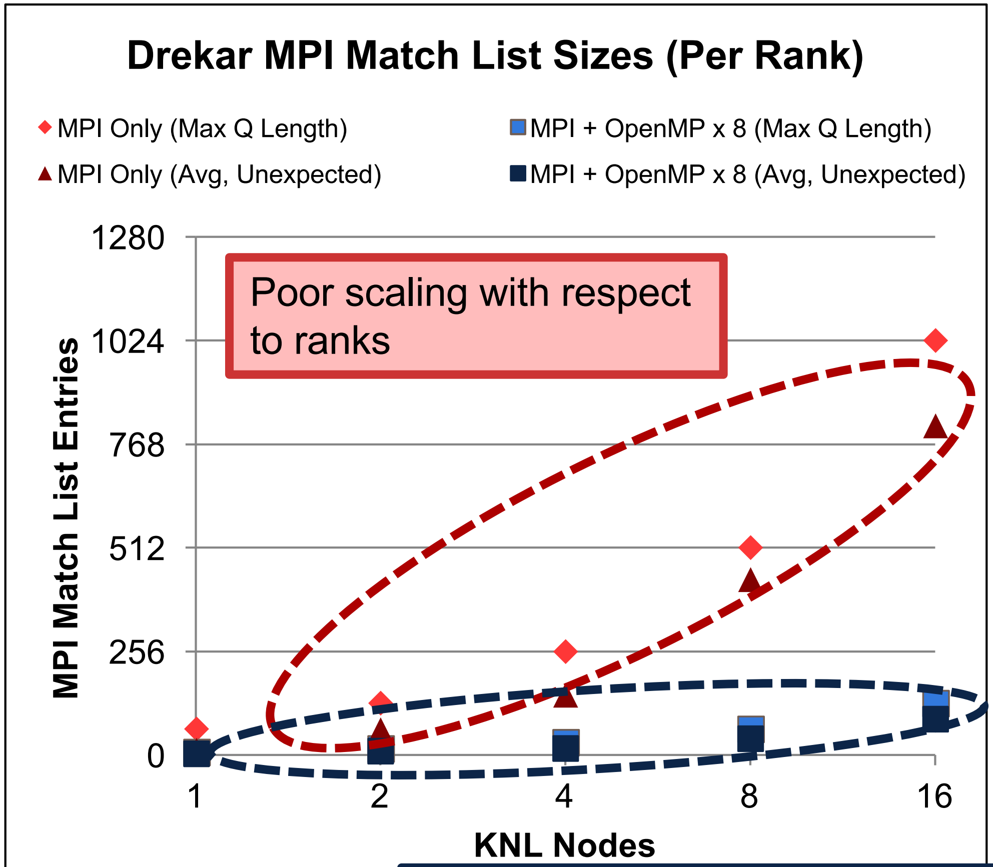
- Codesign aspects:
  - Resource structure sizes on the NIC (how many list entries are required)
  - Are our applications well behaved with respect to resource scaling by MPI ranks?

- Implications for future
  - Want to “right size” the resource entries in future network interconnects/NICs

MPI Receive Queue Analysis



Efficient use of resources



Poor scaling with respect to ranks

Move to hybrid helps reduce demands significantly

Conclusions (from Cray)

- The application data has similar behavior to a broad collection of mini-apps
  - If you take behavior of mini-applications across broad cross-section
  - See lots of cases where mini-apps and applications have extra resource usage on rank-0
- Drekar MPI-only will place significant strain on network interconnect/NIC when used at scale
  - Concern this may cause high cost if the number of receives exceeds NIC resources
- See work from Mike Levenhagen, Ryan Grant etc on deep investigation
  - Extension of OpenMPI so we can perform similar studies on CTS and POWER-EA systems
  - Early results also shared with Cray

Sandia National Laboratories

COMPILER QUALITY/OUTPUT ANALYSIS

Comparing Compiler Technology Generations

Generations of Compilers

- All DOE laboratories receive new/updated compilers at least yearly
  - Improved language support
  - Fixed bugs
  - Better code/instruction generation or new hardware targets
  - Higher level of optimization

- One question is how much better are they getting for our applications in several metrics
  - Time
  - Vectorization
  - Optimization for memory transfers
  - Use of newer hardware functionality

Generations of Compilers

- For Knights Landing these questions come up a lot (especially vectorization)

- Analysis:
  - Study the IC/NALU-CFD Application which is representative of SIERRA STK Mesh use
  - Solves fairly simple "rotor blade" problem using optimized KNL builds
    - Limited OpenMP due to continuing work with Trilinos

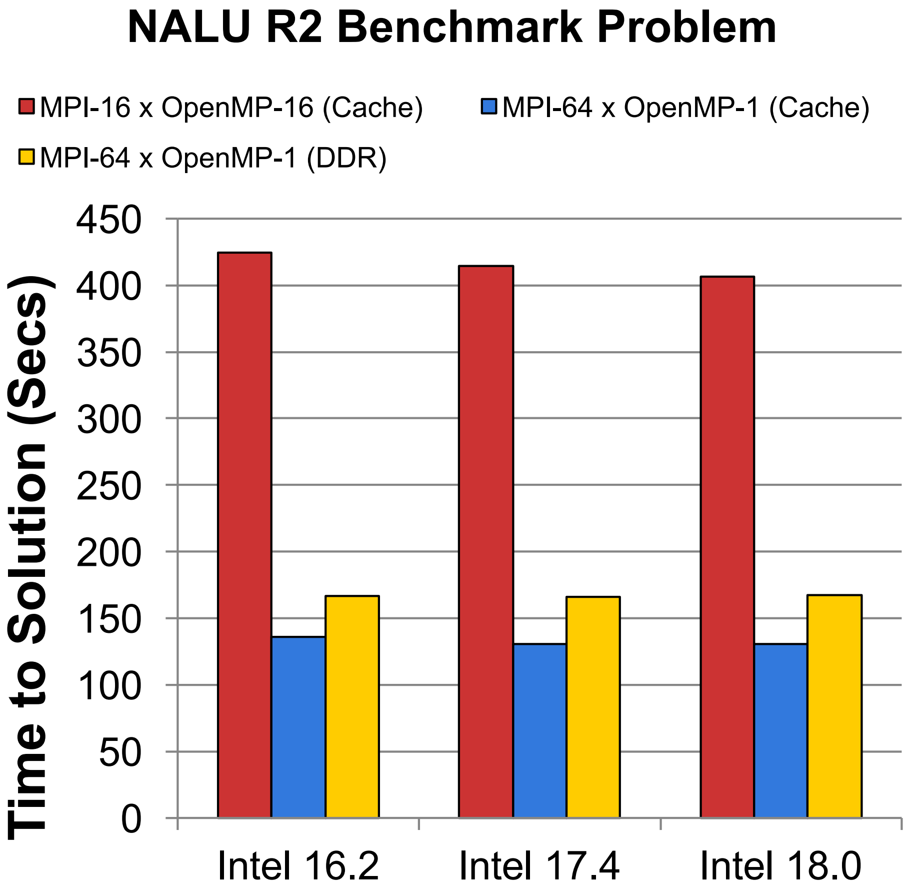
- Used Intel 16.2, 17.4 and 18.0 (Beta) compilers
  - *"Compilers are getting better at producing higher levels of vectorization and code optimization over time – true or false?"*

Execution Time

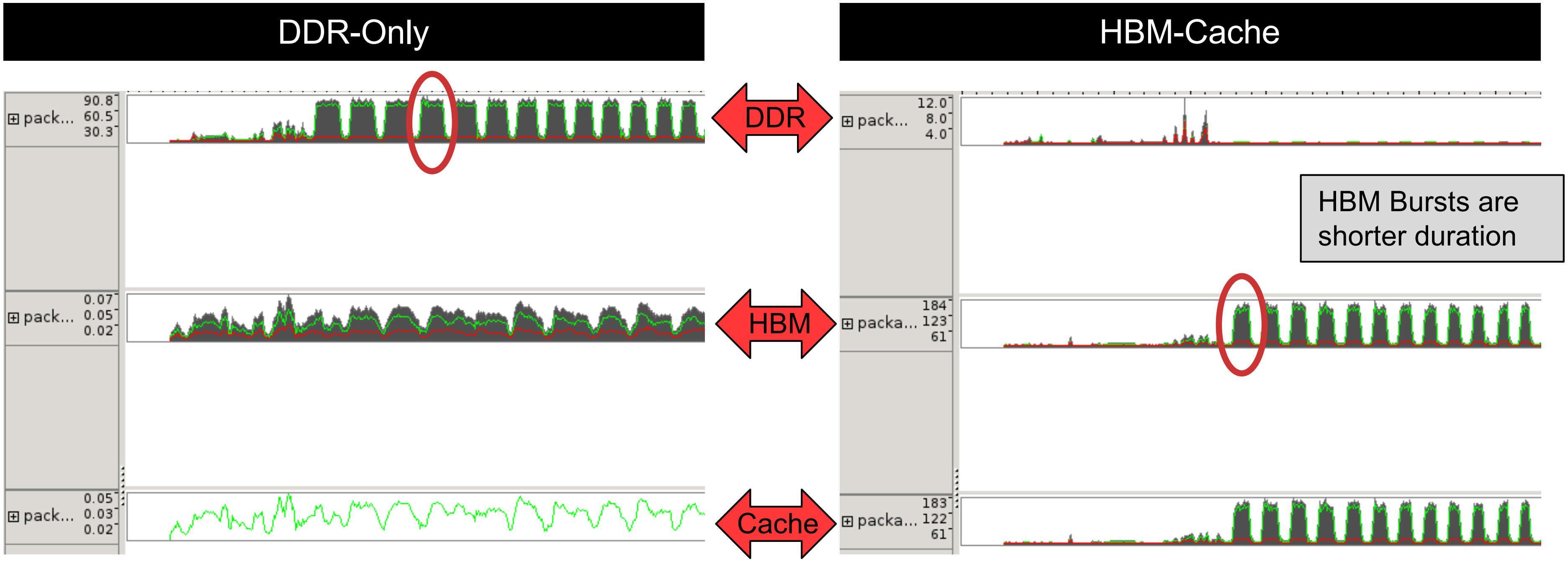
- Intel 18.0 compiler is the fastest (in DDR the results are virtually identical)

- Still see huge differences in threaded versus MPI-only performance because significant number of kernels are not threaded (development on-going)

- Only marginal improvement from HBM/Cache over DDR


NALU R2 Benchmark Problem

Memory Bandwidth Analysis (KNL)



HBM Bursts are shorter duration

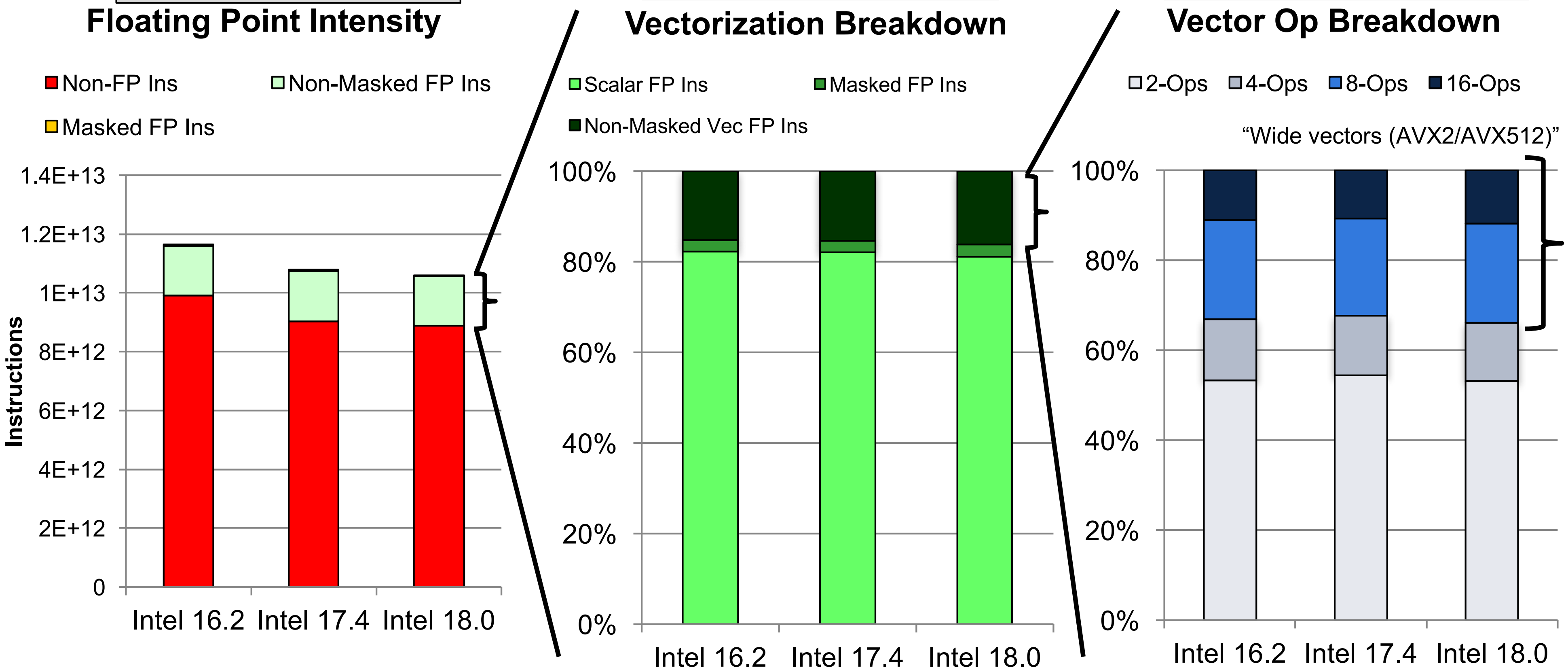
See some improvements in the use of the memory systems but difference is only 2X despite ~5X hardware difference increasing dictated by core performance

Vectorization (Per Rank) (APEX/Tesserae)

Approx 15% is FP Instruction

Approx 20% of FP is vectorized

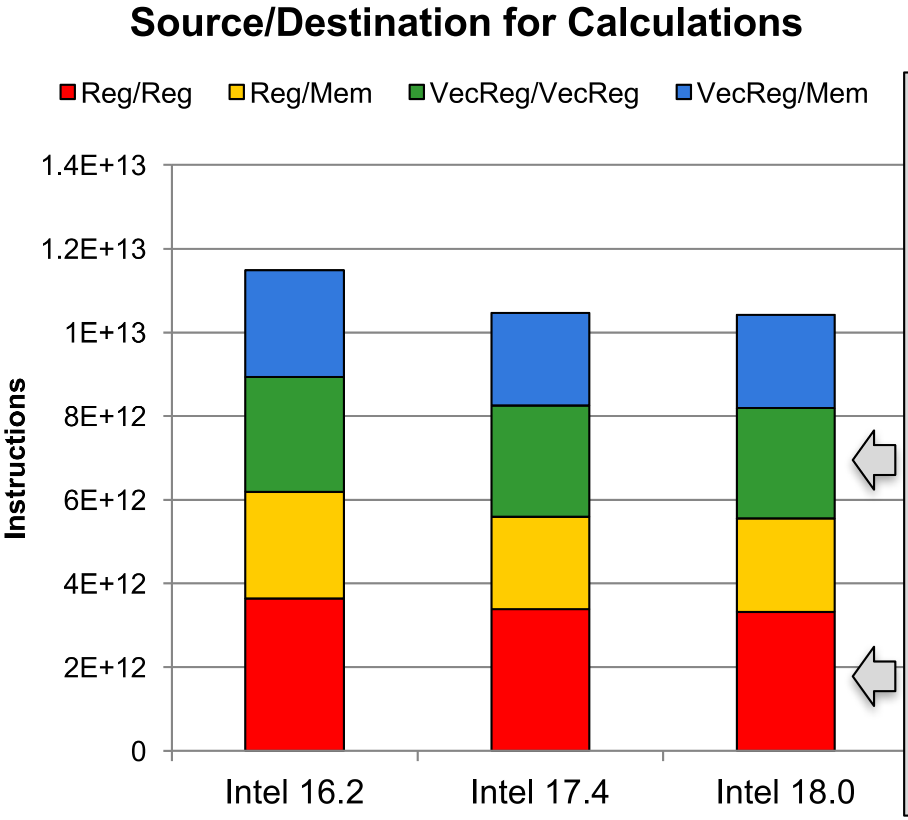
Approx 35% is “wide” vectorized



Fairly significant improvement from 16.2 to 17.4 in instruction count, breakdown similar

Metrics of Interest (Vendors Ask For)

- Memory Transferred/Memory Ref
  - Intel 16.2 – 9.24 Bytes/Ref
  - Intel 17.4 – 9.55 Bytes/Ref
  - Intel 18.0 – 9.72 Bytes/Ref
  - Higher means more efficient transfers
- Bytes to Floating Point Ratio
  - Intel 16.2 – 12.86
  - Intel 17.4 – 11.97
  - Intel 18.0 – 11.72
  - Lower means less data movement



Reduce data motion/improve execution speed

Fairly significant improvement from 16.2 to 17.4 in instruction counts, gradually more data using vector registers

Discussion

- Seeing improvements in time with successive compiler generations over the same code base
  - Shows there is value in pursuing updating compiler versions (even though revalidation is time consuming)
  - Allows us to more efficiently use hardware resources which we have paid for

- Vectorization levels are still lower than we would like even on a system like Knights Landing where the VPUs are very efficient
  - C++ continues to be our nemesis with vectorization
  - Kokkos adding to the complexity with inlining and use of views that cannot pass alias-analysis when used in lambda functions

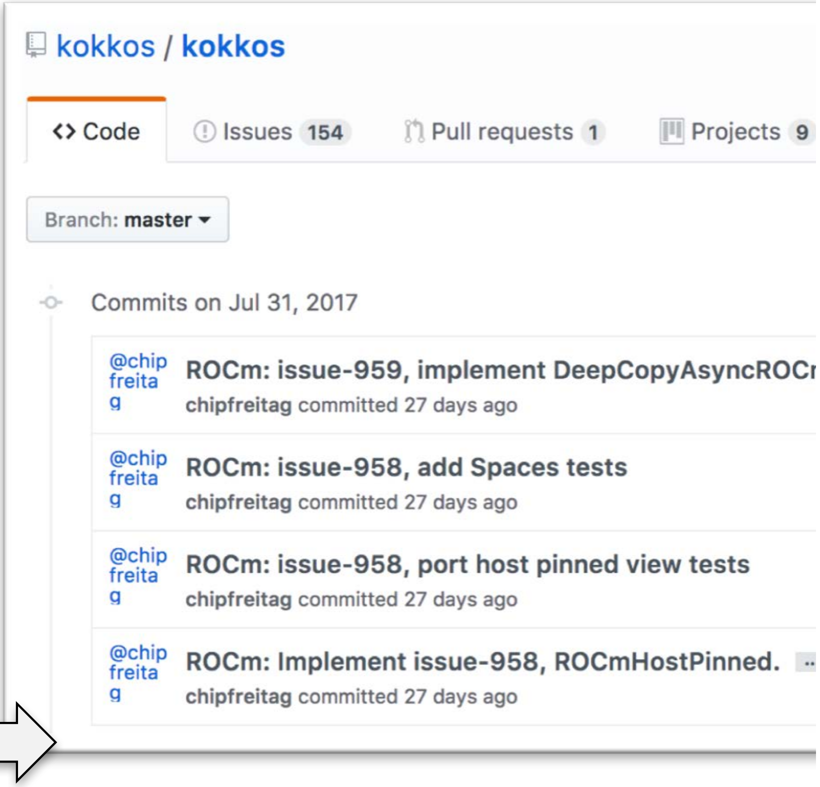
- Future hardware systems with greater memory bandwidth are likely to place much greater emphasis on instruction efficiency/optimization of computation

Sandia National Laboratories

FY17 CODESIGN UPDATE FROM SANDIA

Codesign Highlights for FY17

- Continue to bring up of ATDM codes on Haswell, Knights, POWER8 and GPUs
  - Seeing significant efforts with underpinning of Trilinos and some really big improvements now underway
- Exceptionally broad testing of compilers including LLVM, GCC, IBM-XL, NVCC, Intel, PGI and in some limited cases Cray
  - Early Intel 18.0 working very well for code portfolio
  - 30% reduction in compile times with IBM XL (still takes a long time)
  - Submitted big pile of bugs this year on early releases
- AMD has supplied a ROCm back-end developed for Kokkos (and lots of fixes)
  - Benchmarking on Fiji GPUs underway



Commits on Jul 31, 2017

@chipfreitag ROCm: issue-959, implement DeepCopyAsyncROC
chipfreitag committed 27 days ago

@chipfreitag ROCm: issue-958, add Spaces tests
chipfreitag committed 27 days ago

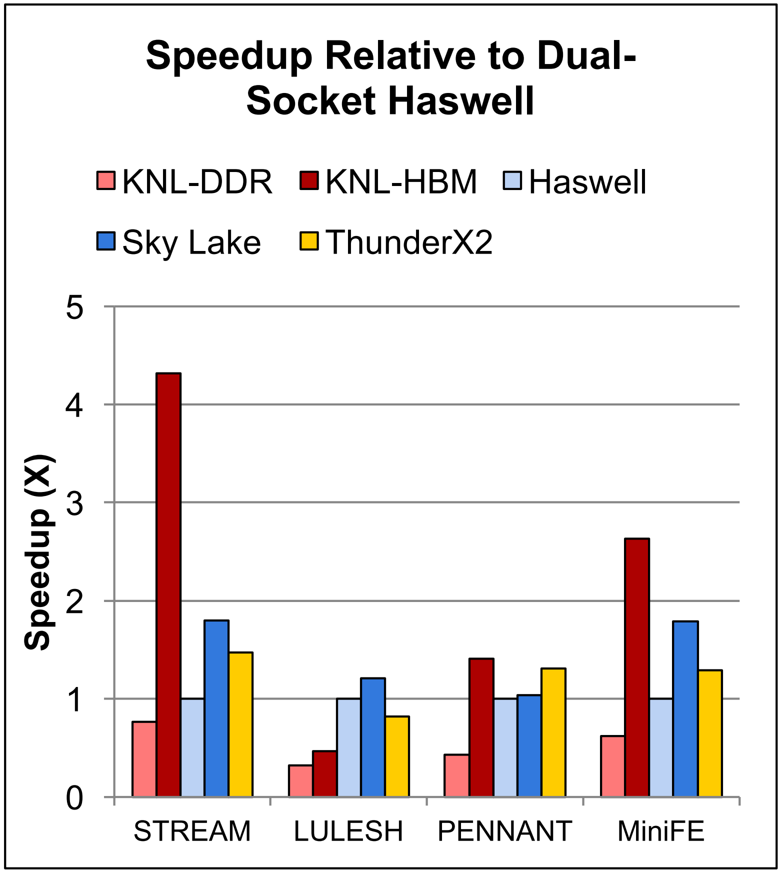
@chipfreitag ROCm: issue-958, port host pinned view tests
chipfreitag committed 27 days ago

@chipfreitag ROCm: Implement issue-958, ROCmHostPinned.
chipfreitag committed 27 days ago

https://github.com/kokkos/kokkos

Codesign Highlights for FY17

- Local team performed early runs on Sky Lake Platinum Xeon
  - Results are looking very strong for many benchmark areas
- Wave-1 shake-down of Cavium ARM processors completed
  - Have NALU-CFD and large suite of packages running (even with the correct answer!), results provided back to HPE
- Continuous Performance Monitoring LDMS System (Jim Brandt, Jeanine Cook and Tom Tucker)
  - Slowly taking shape, prototypes on Mutrino HSW/KNL now underway and data gathering shown when used in DAT modes
  - Hoping for some of this to come online in FY18



Thank You!

- Sandia Application Performance Team ("APT")
- ASC Advanced Test Bed Admin/Support Team
- SIERRA, RAMSES and CTH Code Groups
- Trilinos Developers
- ATDM Code Groups
- ATDM Software Environment Team

- L2 Milestone Review Team!

The FY17 Award for Patience Goes To..

- Clay Hughes

- For services to getting the IBM XL compiler to compile Trilinos and working with the NVIDIA Profiler despite it taking a significant amount of time

Sandia
National
Laboratories

APPLICATION METRIC GATHERING

Application Metric Gathering

- ATDM/SPARC
  - MPI Operation Breakdown
  - MPI Function Times
  - Thread/Rank Scaling
  - Native counter timing comparison (Haswell vs. Knights Landing)
  - Comparison of native solvers vs. Trilinos

- ATDM/EMPIRE/Drekar
  - Native counter timing comparison of Haswell, Knights Landing and POWER w/ P100 GPUs
  - GPU/CPU kernel profiling

- IC/NALU
  - Native counter
  - MPI Match Lists (early in presentation)

Performance Analysis of ATDM/EMPIRE/Drekar

Work by Paul Lin

EMPIRE/Drekar Solver Performance

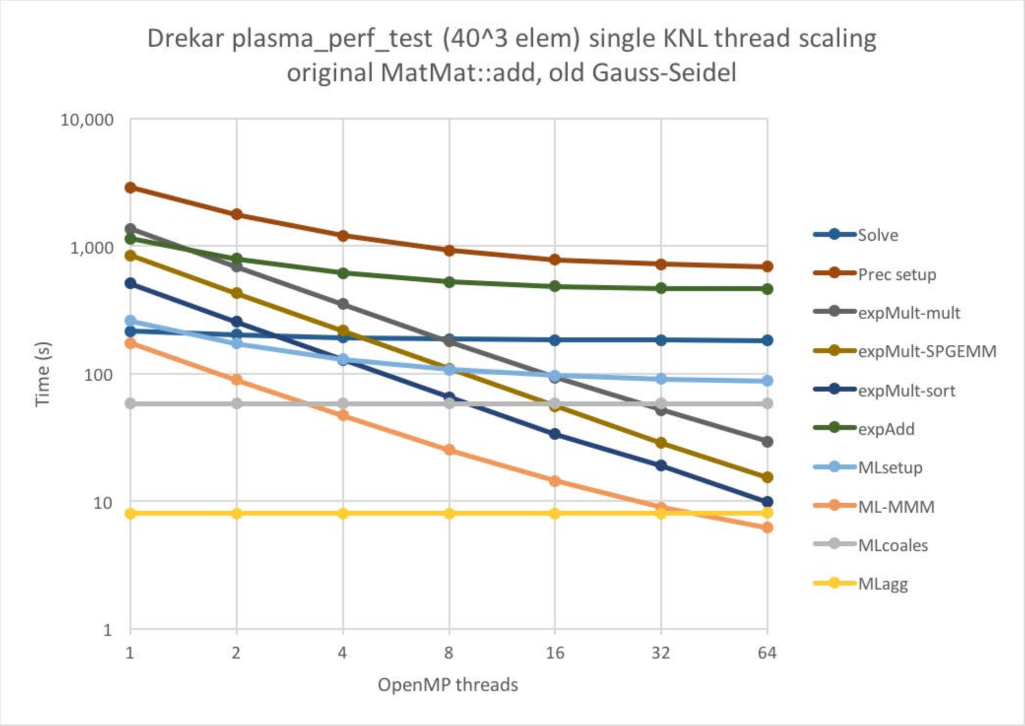
- Focus was on trying to improve performance of ATDM/EMPIRE/Drekar/fluids solve (Trilinos solvers)
- What application developers (e.g. Drekar dev) really want
  - profile code performance, e.g. provide breakdown where time is being spent with the goal of identifying performance bottlenecks
  - even better: help/work with them or TPL developers to improve performance
- Performance profiling and improving thread scaling on KNL
  - mainly focused on single KNL (mutrino/tr2, ellis/bowman) because if doesn't thread scale on single KNL, will not on multiple KNLs
  - after get good single KNL performance, then will focus on multiple KNLs
  - But did look at performance of multiple KNLs (up to 512)
  - EMPIRE lead provided representative plasma test case
- Performance profiling for large-scale simulations
  - Focus on MPI performance
  - Most of this work was for CPUs rather than KNL
  - Employed a different test case (MHD)

EMPIRE/Drekar Thread Scaling on KNL

- Accomplishments
  - Tpetra matrix-matrix addition (worst thread scaling kernel)
    - Significant reduction in time and significant improvement in thread scaling for for single KNL with one MPI process
    - Still in progress: multiple MPI processes and multiple KNLs
  - Gauss-Seidel (second worst thread scaling kernel)
    - Significant improvement in thread scaling for solve (apply GS)
    - KokkosKernels team working to improve setup time for GS
- Had identified additional major kernels that did not thread scale (work still in progress)
  - few items in MueLu setup
  - Tracking all above issues with github issue

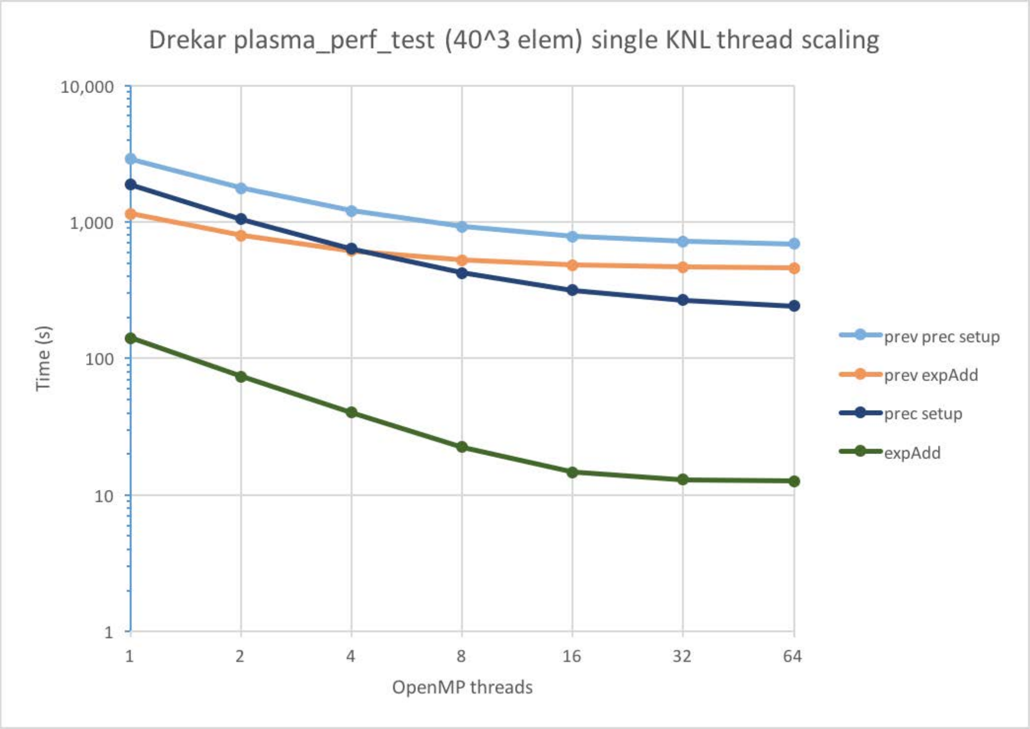
EMPIRE/Drekar Solvers on KNL

- Single mutrino KNL, 1 MPI, increase OMP threads from 1 to 64
- Setup: most expensive kernel for 1 OMP is mat-mat multiply, but KKMEM thread scales well (1 OMP ~47% of setup, 64 OMP ~4% of setup)
- Worst thread scaling kernels:
  1. Tpetra matrix-matrix addition during setup (separate from multigrid)
  2. Application of Gauss-Seidel smoother during solve
  3. A few kernels in MueLu multigrid setup
     - For 1 and 64 OMP, multigrid setup ~9% and ~13% of total setup, respectively



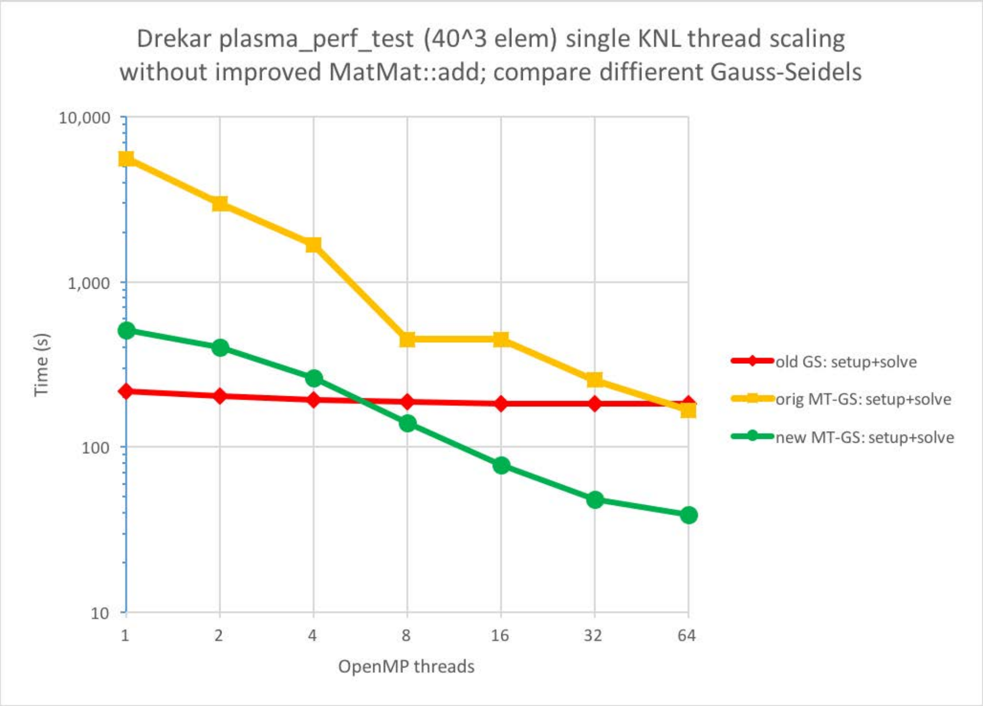
On-Node Improvement (EMPIRE/Drekar)

- Single mutrino KNL, 1 MPI, increase OMP threads from 1 to 64
- Significant reduction in time and significant improvement in thread scaling for for single KNL with one MPI process
- Still in progress: multiple MPI processes and multiple KNLs



Improv. to Gauss Seidel Setup/Apply

- Single mutrino KNL, 1 MPI, increase OMP threads from 1 to 64
- previously used non-threaded Gauss-Seidel (“old GS”)
- switched to KokkosKernels threaded Gauss-Seidel (“orig MT-GS”)
  - initially, threaded Gauss-Seidel setup time was very high, worked with KokkosKernels team (Deveci) to significantly reduce
- Improved MT Gauss-Seidel (“new MT-GS”): good threaded performance for setup to 8 OMP, reasonable threaded solve
  - KokkosKernels team currently working on more improvements



EMPIRE/Drekar Large Scale Runs

- Focused on MPI performance and scaling rather than threaded
- Accomplishments:
  - panzer scaling bug after change from stk_classic to stk (identified and now fixed)
    - Caused a major memory explosion, Drekar was failing with 32k MPI processes when it had previously been run with 1.6M MPI
  - STK CommSparse scaling bug (identified and fixed)
    - Big performance issue at scale; both memory and time explosion
      - same STK bug and Cray MPI bug that caused Nalu to fail the Trinity Phase 2 acceptance test; I was actually simultaneously chasing down this issue with both Drekar and Nalu, not realizing at that time it was the same bugs
  - MueLu setup
    - identified and work with MueLu team to fix a modest scaling issue, for 524,288 MPI reduced setup by ~10%
    - we worked with Zoltan2 team (M. Deveci) on a scaling issue---preliminary results on 131,072 MPI on BG/Q showed a reduction in setup of ~30%
- Identified other issues (work still in progress)
  - MueLu setup
    - Few other MueLu setup scaling issues (working with MueLu team to resolve)
    - Tpetra matrix-matrix multiplication
  - Scaling issues with periodic BCs (panzer scaling issue; pushed back to FY18)

EMPIRE/DREKAR ANALYSIS ON POWER-EA

Work by Clay Hughes

EMPIRE/Drekar on POWER-EA

- GCC + CUDA Builds working successfully although problem sizes often need to be reduced to fit into GPU memory
  - Finding that scratch arrays used in inner kernels are growing as kernel complexity increases

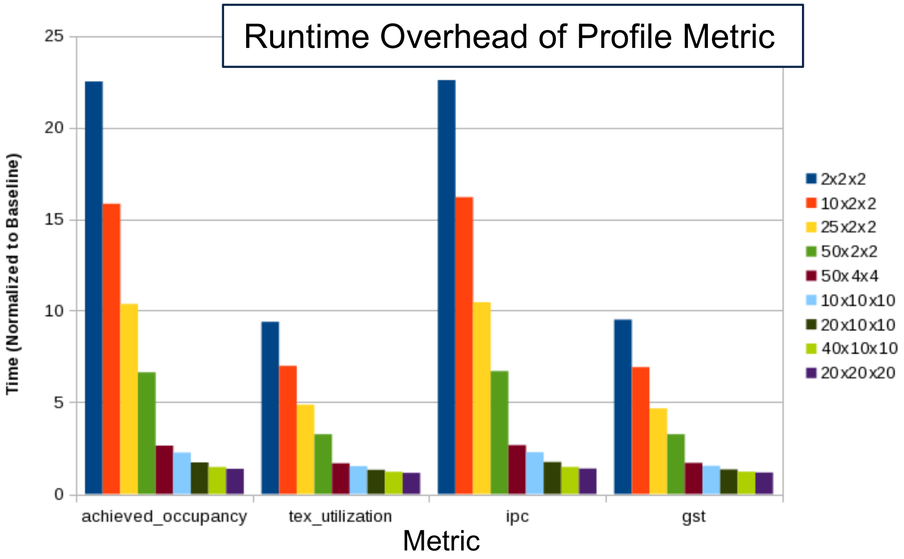
- Significant amount of work trying to get IBM XL to compile code so we can perform a cross-comparison (performance and functionality)
  - Latest IBM XL 14.1.0 does not work correctly
  - Either does not compile the code (errors in particular with OpenBLAS inline assembly for VSX)
  - Or compiles and has a segmentation fault in BLAS routines (not present in GCC version of the code)

- Metric analysis of EMPIRE/Drekar on Pascal GPUs an issue

Performing Metric Gathering on P8-EA

- Gathering metrics using NVIDIA profiler in POWER-EA systems is causing significant increases in runtime

- Specific metrics causing problems:
  - DRAM Read/Write Throughput
  - L2 Read/Write Throughout
  - GLD
- In some cases the runtimes exceed 48 hours which are our job queue limits
  - The worst profiling overhead we have seen in tools



Outcomes

- Profiling overhead bug is reported to NVIDIA
  - Bug: #1977033, current status is open, NVIDIA need small reproducer
  - Not sure that small reproducer will show the outcome so we are looking at whether SDK samples will induce the behavior
- Potential IBM XL Inline Assembly Bug in XL 14.1.0 Beta
  - Reported to IBM Toronto compiler team
  - Investigation underway
- Feedback to EMPIRE/Drekar development on the use of large “work sets” which can significantly bloat memory consumption on GPU

ANALYSIS OF ATDM/SPARC

Work by Courtenay Vaughan

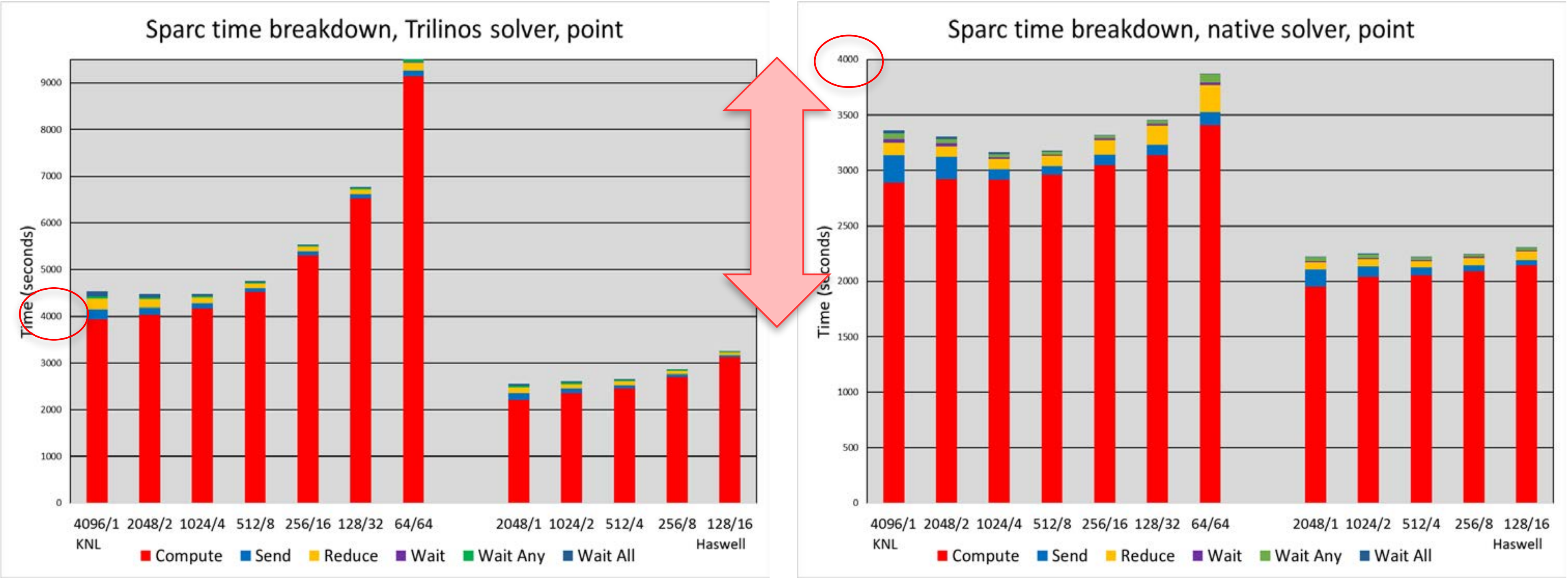
SPARC Analysis

- Performed for the Trinity Phase-II open-campaign to assess scaling

- SPARC is written to utilize two approaches:
  - Native Solver written in OpenMP as prototype (considered to be very fast)
  - Full Trilinos solvers being slowly ported to Kokkos (many packages, slower but being optim'd)

- Questions:
  - What needs most improvement?
  - What packages?
  - How is scaling on and between nodes performing today?

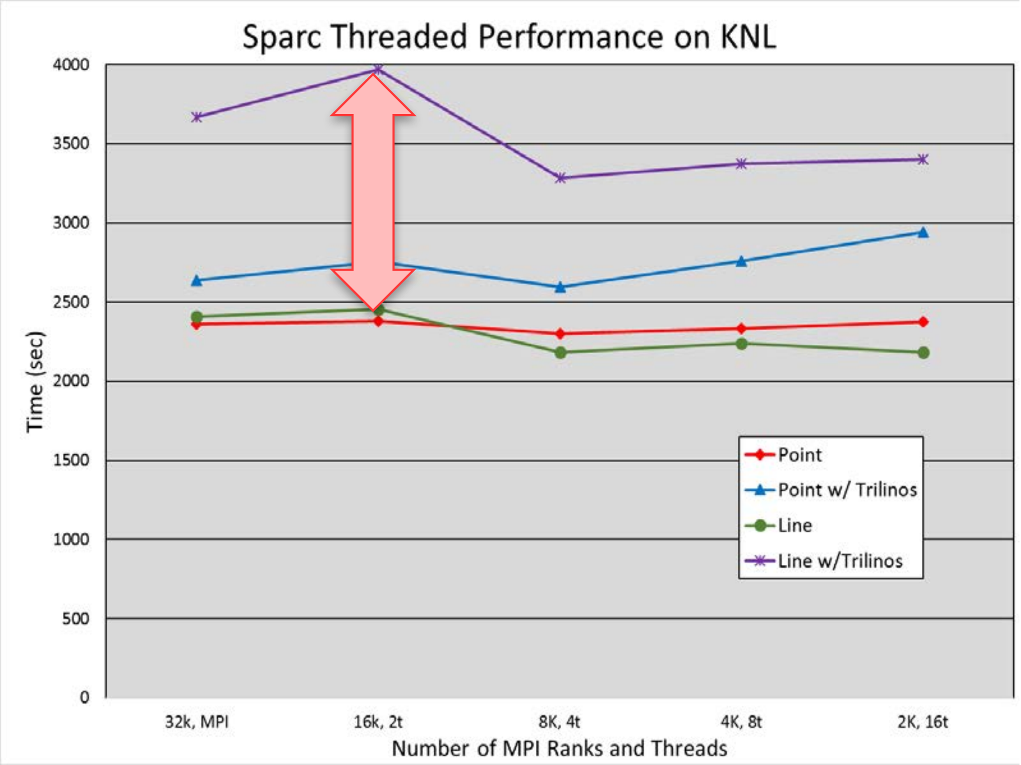
SPARC Runtime with Solver Variants



Up to 2X performance difference (native faster) but .. MPI times are slightly faster in Trilinos (shows history as *MPI-only code*, SPARC shows what a *newly* threaded, from group up code *can do*.

Thread Scaling of SPARC on KNL

- Captured with in-code timings but also checked against CrayPAT outputs (for small nodes)
- Trilinos Line Solver is approx. 50% slower
- Trilinos Point Solver is competitive but does not scale as well with large thread counts



Observations from MPI Analysis

- Number of messages exchanged decreases from 130000 to 103000 per MPI rank when going from MPI only to 64 threads
  - So the total number of messages decreases substantially
- The bytes exchanged grows from 1.2 E10 to 1.46 E11 bytes per MPI rank when going from MPI only to 64 threads
  - The total message traffic drops from 4.5 E13 to 9.33 E12 bytes
  - The average message size goes from 82 KB to 1.3 MB
- For the native solver, the number of reduces stays constant at 47500 calls and 3.6 MB
- For the Trilinos solver, the number of reduces stays constant at 30000 calls and 2.4 MB

Update from SPARC Team

- Andrew Bradley has been developing a faster solver for Trilinos (using experiments from native solver)
  - Early results which were used for the Power API L2 show significant improvement
  - Now as fast/faster than the native solver
  - Builds on experiences from native development but with additional optimizations

- Will be pushed into Trilinos before the end of the FY

LMDS Kernel and Continuous Performance Tracing

Kernel Monitoring Work by Si Hammond, Christian Trott, Tom Tucker, Jim Brandt and Ann Gentile

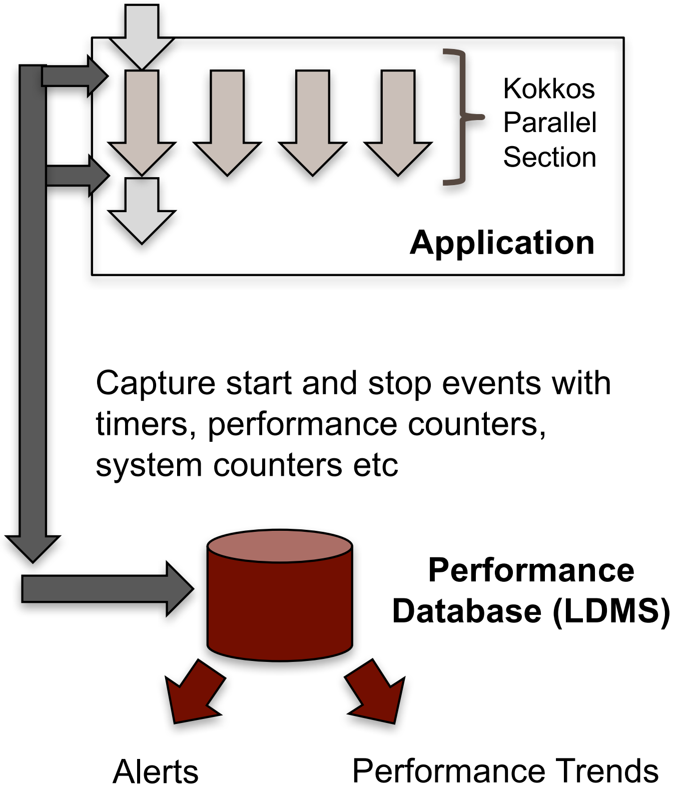
Continuous Monitoring Work by Jeanine Cook, Jim Brandt, Ann Gentile and Tom Tucker

Application Kernel Monitoring

- KokkosP-based kernel profiling direct into LDMS metrics database
  - Demonstrated on KNL, Haswell and POWER8
  - Ability to hook into Kokkos kernels and regions without requiring changes to compiler, linking etc

- Provides detailed drill down for Kokkos-enabled applications
  - Can connect with vendor tools, PAPI, perf, etc
  - Data movement (e.g. NVLINK, MC-DRAM etc)
  - Provides Kokkos context (e.g. “parallel-for”)

KokkosP-Enabled Monitoring



Kokkos Parallel Section

Application

Capture start and stop events with timers, performance counters, system counters etc

Performance Database (LDMS)

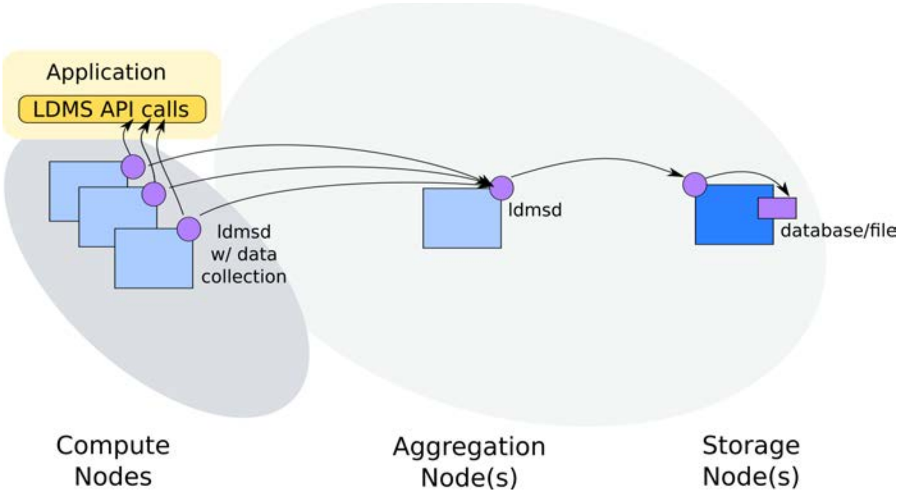
Alerts

Performance Trends

Work by Si Hammond, Christian Trott, Tom Tucker, Jim Brandt and Ann Gentile

LDMS Continuous Monitoring

- Continuous HPC system monitoring framework that:
  - Collects numeric data
  - Moves and aggregates data
  - Stores data
  - Analyze data
    - Troubleshooting
    - Optimization
    - Inform future systems
- Plug-in architecture
  - Customized samplers for data collection
- Backend
  - Storage (SOS DB)
  - Analysis (Python)
  - Visualization (Grafana)
- Application-based samplers
  - PAPI, Perf, MPI, DLoad

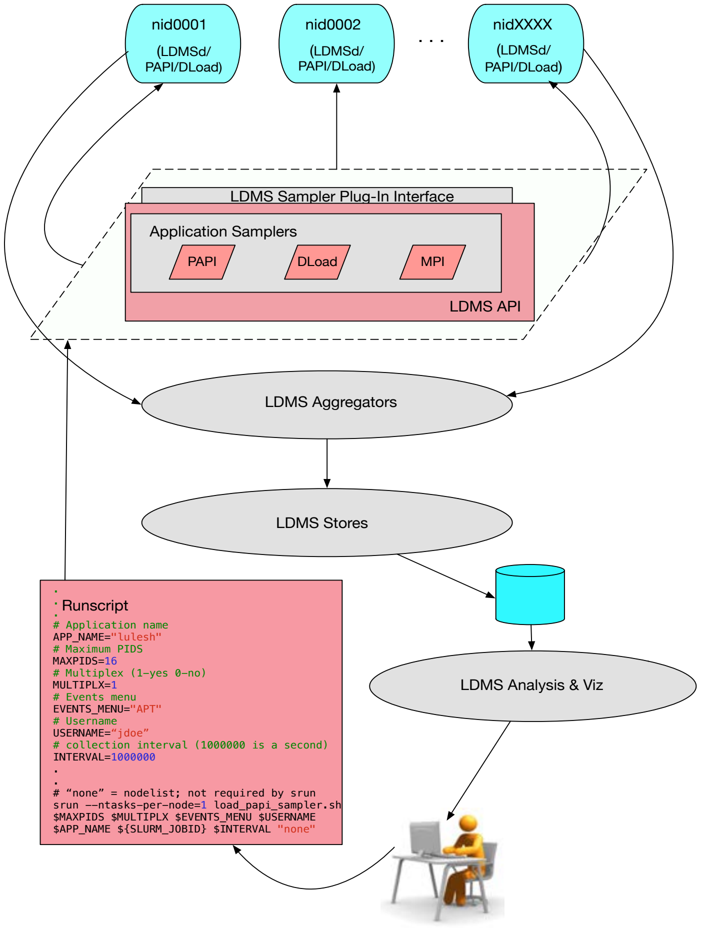


* Only the current data is retained on-node

Work by Jeanine Cook, Tom Tucker, Jim Brandt and Ann Gentile

Application LDMS “Samplers”

- PAPI
  - Menu-driven
    - Instruction mix (vectorization, cycles per instruction (CPI))
    - Branch mispredict rates
    - Cache/memory statistics (miss/hit rates)
    - Average (estimated) memory bandwidth
    - Stall cycle profiles (which components incur stalls)
    - Other events for performance characterization work
  - Cross-platform
    - Haswell/Broadwell, Ivybridge
    - Very near future: Power8, KNL
  - Opt-in and configure through runscript variables
- DLoad
  - Enables dynamic loading and reconfiguring (e.g., application/PAPI events) of application samplers
- MPI
  - Developed by UCF
  - Working on usability issues
- Perf
  - Complete after PAPI development is complete and tested
  - Currently low priority



Runscript

```
.
.
.
# Application name
APP_NAME="lulesh"
# Maximum PIDS
MAXPIDS=16
# Multiplex (1-yes 0-no)
MULTIPLX=1
# Events menu
EVENTS_MENU="APT"
# Username
USERNAME=“jdoe”
# collection interval (1000000 is a second)
INTERVAL=1000000
.
.
# “none” = nodelist; not required by srun
srun --ntasks-per-node=1 load_papi_sampler.sh
$MAXPIDS $MULTIPLX $EVENTS_MENU $USERNAME
$APP_NAME ${SLURM_JOBID} $INTERVAL "none"
```

LDMS Status

- PAPI & DLoad samplers
  - Small-scale testing complete on several testbeds
  - Large-scale testing on Mutrino/Haswell scheduled 08/31
  - All KNL testing by 11/01
- MPI sampler
  - Targeting testing completion by Sept 15
- LDMS w/Application Samplers, and Backend, Analysis, basic Viz
  - Plan to launch on Mutrino/Haswell by Sept. 15
- Working with Sierra team to integrate LDMS application monitoring into nightly testing
  - Hope to complete integration and testing by 12/01

Sandia
National
Laboratories